

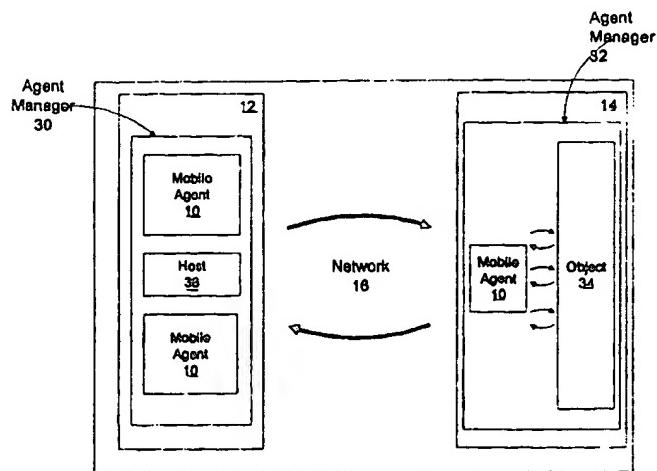
PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION  
International Bureau

## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>6</sup> :  G06F 13/00		A1	(11) International Publication Number:  WO 98/21662
			(43) International Publication Date:  22 May 1998 (22.05.98)
(21) International Application Number: PCT/US97/20232 (22) International Filing Date: 12 November 1997 (12.11.97)  (30) Priority Data: 60/030,906 14 November 1996 (14.11.96) US		(81) Designated States: AU, CA, CN, IL, JP, KR, MX, NO, NZ, SG, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).  <b>Published</b> <i>With international search report.  Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>	
(71) Applicant: MITSUBISHI ELECTRIC INFORMATION TECHNOLOGY CENTER AMERICA, INC. [US/US]; 201 Broadway, Cambridge, MA 02139 (US). (72) Inventor: WALSH, Thomas, C.; 47 Tileston Street #2, Boston, MA 02113 (US). (74) Agent: TENDLER, Robert, K.; 65 Atlantic Avenue, Boston, MA 02110 (US).			

(54) Title: ITINERARY BASED AGENT MOBILITY INCLUDING MOBILITY OF EXECUTABLE CODE



## (57) Abstract

In accordance with the present invention, a mobile agent object (10) executes a first method (18) on a first computer (12), migrates from the first computer (12) to a second computer (14), and executes a second method (20) on the second computer (14). The first and second methods and first and second computers are designated in an itinerary (28). The agent includes both data (22) and executable code (24, 26) which are serialized for transmission from the first computer to the second computer as data. The data containing the agent is then deserialized in the second computer to regenerate the agent in the form of an object. The executable code portion of the agent can be supplemented with code from a home codebase located on another computer.

**FOR THE PURPOSES OF INFORMATION ONLY**

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AI	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

5

TITLE OF THE INVENTION  
ITINERARY BASED AGENT MOBILITY INCLUDING MOBILITY OF  
EXECUTABLE CODE

10

## CROSS REFERENCE TO RELATED APPLICATIONS

A claim of priority is made to U.S. provisional patent application Serial Number 60/030,906, entitled USE AND COLLABORATION OF MOBILE AGENTS IN A COMPUTER NETWORK, filed November 14, 1996.

15

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR  
DEVELOPMENT  
Not applicable

20

## BACKGROUND OF THE INVENTION

The present invention is generally related to network computing, and more particularly to mobile objects.

25

Mobile objects that are transmitted across a computer network are known. Mobile objects are comprised of code and data, both of which are transmitted across the computer network. Technologies such as client-server protocols, remote procedure call protocols, and distributed object technologies are employed to transmit mobile objects across the computer network. These technologies implement either a "push" model or a "pull" model, both of which have drawbacks.

30

In a "pull" model the code for an executing object is downloaded from a network source such as a web server. When a particular portion of code becomes necessary for operation, that portion of code is first sought on the local filesystem. If the local filesystem does not include that portion of the code, a network request is sent to another computer in the network in order to prompt transmission of that portion of code to the local computer. However, as different classes of sub-objects are created, a separate network request must

35

-2-

be sent to retrieve code for each class. In the case of a Hyper-Text Transfer Protocol ("HTTP") request, each request must re-establish a Transmission Control Protocol/Internet Protocol ("TCP/IP") socket connection to the web server. 5 Thus, when downloading a large number of classes, a large number of connections must be established and a large amount of network overhead is incurred.

Another drawback associated with the pull model is that as an object travels, the code which enables the object to operate must be re-downloaded at each computer. More 10 particularly, when the object arrives at each a destination computer, the object downloads any code which is required but not present on the local filesystem, even if that code was downloaded at the previous destination computer. Previously 15 downloaded code is not cached. Network overhead is incurred as a consequence.

In a "push" model the code for an executing object is carried with the mobile object. Prior to launching the mobile object, all of the code that will be needed by the 20 object is identified. The code is packaged with the agent and pushed around the network to each destination computer. While the push model reduces network requests in comparison 25 with the pull model, network overhead is still incurred because of the relatively large amount of code that is pushed to every destination computer. In some cases the object will push code for classes that are no longer needed. For example, an agent may create an instance of a particular 30 class of object only in very exceptional circumstances. In such a case it is inefficient for the object to push the code for this class.

Another limitation of known mobile objects is difficulty in examining and predicting destination information. Known mobile objects initiate travel by performing a subroutine call. The method is sometimes given a name such as "go" or "moveTo," and the caller is responsible for specifying a name 35 or Uniform Resource Locator ("URL") indicating the destination computer for the mobile object. When the travel method is called, the execution of the mobile object is

- 3 -

halted and the mobile object is converted into a network transmittable form. On the destination computer the mobile object is restored from the network form and restarted. This restart can either occur on the instruction directly following the call to the travel method, or at the beginning  
5 of some known method.

#### BRIEF SUMMARY OF THE INVENTION

In order to provide efficient mobility of code for a  
10 mobile agent object ("agent"), a "mobile codebase" object is constructed. The mobile codebase travels with the agent and serves as a repository of code for classes that facilitate agent operation. When the agent is launched, the launcher specifies a list of related classes. These classes are  
15 retained in the mobile codebase. The launcher also specifies a Uniform Resource Locator ("URL") that points to a network source ("home codebase") from which code can be downloaded. If the agent is required to instantiate an object comprising  
20 code that is not in the mobile codebase, a network request is sent to the home codebase to retrieve that code. Code that is downloaded in this manner is retained in the mobile codebase.

The combination of the mobile codebase and home codebase  
25 improve agent operation and reduce network overhead. In a preferred embodiment an agent is launched with a mobile codebase that is preloaded with classes which are certain to be needed. Classes that are unlikely to be needed are stored in the home codebase. Thus the agent with mobile ccdebase and home codebase provide the desirable features of the push  
30 and pull models without the undesirable network overhead incurred by transmitting unneeded classes around the network or re-downloading code.

Agent travels are defined by an itinerary data structure. The itinerary is composed of a list of destinations, each of which includes: (1) the hostname of a destination computer on the network to which the agent should travel, and (2) the name of a method that the agent should  
35

-4-

execute when the agent arrives at the destination computer. An agent is allowed to modify its itinerary. When an agent modifies its itinerary, the agent owner is automatically notified, thus allowing the owner to "follow" the agent wherever it may travel.

The itinerary provides advantages related to agent and network management. Inspection of the agent itinerary reveals where the agent has travelled and where the agent may travel in the future. Since the itinerary is a standard data structure, an observer can depend on it to be present. Further, since the itinerary is a separate data structure from the agent, the observer does not need special knowledge of the internal structure of the agent in order to examine the itinerary.

15

#### BRIEF DESCRIPTION OF THE DRAWING

The invention will be more fully understood from the following Detailed Description of the Invention, in conjunction with the Drawing, of which:

Fig. 1 is a block diagram which illustrates agent migration;

Fig. 2 is a block diagram which illustrates agent execution in conjunction with agent migration;

Fig. 3 is a block diagram of an agent;

Fig. 4 is a block diagram of agent interaction with agent managers on multiple computers in a network;

Fig. 5 is a flow chart which illustrates agent mobility;

Fig. 6 is a block diagram which illustrates agent operation according to an itinerary;

Fig. 7 is a block diagram which illustrates the agent runtime environment; and

Fig. 8 is a flow chart which illustrates agent deserialization.

35

#### DETAILED DESCRIPTION OF THE INVENTION

Referring to Fig. 1, an agent 10 comprises a mobile

-5-

object which can travel from a first computer 12 to a second computer 14 in a computer network 16. The agent is generated and stored in memory on the first computer for a specific purpose, and includes both data and executable code. The  
5 agent executes on the first computer and then migrates to the second computer by being transmitted across the network. The agent then resumes execution on the second computer. When the agent completes execution on the second computer, the agent acts in accordance with an agent itinerary. For example, the agent will migrate to another computer if the  
10 itinerary so indicates.

Referring now to Figs. 1 and 2, the agent executes a sequence of instructions during operation. In particular, a first method 18 is executed on the first computer 12 and a second method 20 is executed on the second computer 14. As a result, migration between computers is transparent from  
15 the perspective of the agent.

Referring to Fig. 4, agent mobility is preferably facilitated through use of JAVA agent manager Objects 32 and an Object Serialization facility (a product of Sun Microsystems which is known in the art). Object Serialization operates to serialize objects such that a representative stream of bytes, i.e., data, is generated therefrom. Object Serialization facilitates transmission of an agent across the network by serializing the agent 10 into a format that is suitable for network transmission. Serialization of the agent may also include serialization of sub-objects which are contained within member variables of the agent. In accordance with the present invention, the  
20 serialization process involves the serialization of executable code, data, and an itinerary as hereinafter discussed in greater detail. Once the agent has been serialized, a communication socket to the second computer is opened and the resulting stream of bytes is transmitted to  
25 the second computer along the socket.

The serialized agent is regenerated in the second computer 14 upon arrival. In particular, the stream of bytes transmitted along the socket is deserialized in the second

-6-

computer. Deserialization regenerates the agent in the form in which the agent existed prior to migration, i.e., an object. Following deserialization, the agent executes a predetermined method specified by its itinerary. The predetermined method may prompt interaction with another object 34 located on the second computer. When the method has been executed, the agent is serialized and transmitted to yet another computer, or possibly back to the first computer. The agent is therein deserialized upon arrival and proceeds to execute another predetermined method and interact with that computer. In the case of the agent migrating back to the first computer, such interaction could be with a host application 36 through which an information query was entered. The agent continues migrating across the network in this manner as execution requires. Eventually, the agent is terminated when execution of the itinerary is complete. It should be noted that the agent need not necessarily migrate to the initial host (first) computer after executing on a remote (second) computer. The agent could travel to multiple computers, and may not return to the initial host computer.

Referring to Fig. 3, each agent may include data 22, code 24, sub-objects 26 and an itinerary 28. The data portion 22 includes internal state information associated with the agent. The code portion 24 includes executable code associated with operation of the agent. The itinerary portion 28 specifies destinations for the agent. Sub-objects 26 include code of distinct objects contained within the agent. The agent carries the code for these sub-objects.

The code 24 portion of the agent 10 includes a mobile codebase and a reference pointer to a home codebase. The mobile codebase comprises executable code which is retained as part of the agent. When the agent migrates, the mobile codebase is transmitted along with the data, sub-objects and itinerary to the destination. The home codebase is executable code which resides on the computer which originally created and launched the agent or a suitable network server (such as a web server). The reference pointer

-7-

5

to the home codebase is a Uniform Resource Locator ("URL") that allows remote access to the home codebase from other computers in the network. If the agent requires code that is not present in the mobile codebase during execution, the agent retrieves the required code from the home codebase and retains such retrieved code in the mobile codebase.

10

Referring to Figs. 3 and 6, the itinerary 28 is a data structure which defines agent migration parameters. The itinerary is composed of a list of destinations and the name of a method associated with each such destination. In a preferred embodiment, each destination entry contains the Transmission Control Protocol/Internet Protocol ("TCP/IP") host name of a computer on the network to which the agent is designated to migrate. The method associated with the respective destination is invoked upon arrival at that destination. Hence, each destination represents a location and a task to be performed at the location.

15

Figs. 4 and 5 illustrate agent mobility. When the determination is made that the agent should migrate, the agent manager 30 examines the itinerary 28 to ascertain the next destination to which the agent 10 is designated to migrate. The agent manager 30 establishes a network connection to the agent manager 32 at the destination computer 14. The agent manager 30 passes the agent and related travel information to the destination agent manager 32. The agent manager 30 then deletes the local copy of the agent, terminates any execution threads which were being used by the agent, and performs other appropriate clean up activities.

20

From the perspective of the destination computer, the agent arrives at the destination computer as illustrated in step 38. As previously described, the agent arrives in the form of data which includes the code, sub-objects, itinerary and other data. Java/Object Serialization assists the de-serialization and reconstruction of the agent as depicted in step 40. The server then spawns a new thread in which the agent will execute as shown in step 42. A persistent local copy of the agent is then made as illustrated in step 44.

25

-8-

As depicted in step 46 the agent manager invokes the proper method of the agent for execution in the spawned thread at this node. The itinerary contains information indicating which method should be invoked at each designated destination computer. The agent then executes as illustrated in step 50.

5

10

15

20

25

30

35

When the agent has completed execution of the indicated method, another persistent copy of the agent is stored on the local disk as depicted in step 52 in order to prevent redundant execution in the event that the computer malfunctions before the agent migrates. The agent manager then examines the agent itinerary as illustrated in decision step 54. If the itinerary does not indicate any further destinations then agent execution is deemed complete as shown in step 56. However, if the itinerary indicates further destinations, then the agent manager determines whether the next destination is associated with another computer in decision step 58. If the next destination is not another computer, flow returns to step 44, thereby bypassing network traffic. However, if the next destination is associated with a different agent manager then a network connection to the indicated agent manager is established as depicted in step 60. The destination contains a string which indicates where to travel. The string contains either the name (host name) of a computer or a URL to a distributed object. The agent is then serialized as illustrated in Step 62 and subsequently transmitted to the next agent manager as depicted in Step 64.

Migration in accordance with the itinerary 28 is further illustrated in Fig. 6. The itinerary includes a listing including identification of each agent manager 66 to which the agent is configured to migrate and identification of a method 68 to execute at each respective agent manager. The agent migrates through the agent managers listed in the itinerary sequentially in the order in which the servers are listed, invoking each respective method upon arrival. Such method could be, for example, querying or updating a database on the destination computer.

Fig. 7 illustrates the agent runtime environment. Each

-9-

computer 70 on the network includes a Java Virtual Machine 72 running therein. A agent manager 74 runs on each respective Virtual Machine 72. The agent manager includes a plurality of threads 76, each thread having an agent 78 associated therewith. In particular, a new thread is formed when a new agent arrives at the server, and the agent executes on that thread. Security mechanisms prevent each agent from operating outside the scope of the associated thread. When agent execution completes, the agent migrates to the next destination identified in the itinerary.

In order for migration and execution to function properly, the agent manager modifies the mechanism by which Java loads classes. The Java standard library provides an object called a "ClassLoader" which can be subclassed by a programmer to modify the rules that Java follows to load classes. The ClassLoader class allows web browsers to download "applets" from a web server. The agent manager provides a specialized ClassLoader which allows the executable code for mobile agents to travel with the agent.

During a typical execution of a Java program, a Java interpreter loads classes as needed for execution from a logical file system on the local computer. However, agents become separated from the home computer and the home codebase during migration, i.e., agents are separated from the directory or directories on the filesystem of the home computer where the code is stored. Further, sub-objects sometimes need to be created as the agent executes, either as member variables or as temporary variables within the methods. These sub-objects could be instances of classes which are part of the standard Java packages or they could be instances of new classes written by the programmer of the agent. However, since the agent is executing on a different computer than the home computer following migration, the Java virtual machine may not be able to directly retrieve the code for the particular sub-object. To allow agents to properly load classes and construct objects when remote from the home computer, a special ClassLoader is employed by the agent manager whenever an agent attempts to construct an object.

-10-

5       The ClassLoader object first attempts to locate a class on the local file system of the computer. If the class is not on the local file system, the ClassLoader searches the mobile codebase. If such attempt fails, the ClassLoader sends a request back to the home machine of the agent. The agent's travel information contains a reference such as a URL which refers back to the home codebase on the home computer, and by accessing the URL the ClassLoader can retrieve classes from the home computer.

10      Agent deserialization is illustrated in Fig. 8. As depicted in step 80 the serialized agent arrives as a stream of data. Java Development Kit Object Serialization classes then call an agent manager related method to perform deserialization as shown in step 82. The method reads agent related objects from the object stream as depicted in step 84. In particular, the mobile codebase, itinerary and codebase URL are read by the method. A custom Classloader and ObjectInputStream are constructed as shown in step 86. Object Serialization then reads the agent and other objects from the input stream as depicted in step 88 and step 90, respectively. When the related objects are read, the ClassLoader retrieves the classes which comprise the agent as illustrated in step 92.

25      The new object's class is loaded once the classes are retrieved. As depicted in step 94, if the class is determined to have been previously loaded then the previously loaded class instance is employed (step 96) and the object is instantiated using the loaded class object as shown in step 98. However, if the class has not been previously loaded as determined in inquiry step 94, then inquiry is made as to whether the class is on the local class path of the destination computer as illustrated in decision step 100. If the class is present on the local machine then the class is loaded therefrom as shown in step 102 and the object is instantiated using the loaded class object as depicted in step 98. However, if the class is not found on a local class path as shown in step 100 then an inquiry is made as to whether the class is in the mobile codebase as shown in step

-11-

104. If the class is located in the mobile codebase then the  
class is loaded therefrom as shown in step 106 and the object  
is instantiated using the loaded class object as shown in  
step 98. However, if the class is not located in the mobile  
codebase in decision step 104 then the class is loaded by  
5 accessing the home codebase as shown in step 108. If the  
codebase pointer is an Internet type pointer such as a Hyper-  
Text Transfer Protocol ("HTTP") URL, as determined in inquiry  
step 110, then an HTTP request is sent to the local web  
server as shown in step 112 and the retrieved class is added  
10 to the Mobile CodeBase as depicted in step 114. However, if  
the codebase URL is not a HTTP URL then a remote method  
request is sent to a RemoteClassLoader object as shown in  
step 116 and the class is added to the mobile codebase as  
15 shown in step 114. In either case, the object is  
instantiated using the loaded class object as shown in step  
98. If more objects exist in the stream to be retrieved, as  
determined in decision step 118, then another object is read  
from the input stream as illustrated in step 90. If there  
20 are no more objects to be retrieved, flow terminates.

Having described the preferred embodiments of the  
invention, other embodiments which incorporate concepts of  
the invention will now become apparent to one of skill in the  
art. Therefore, the invention should not be viewed as  
25 limited to the disclosed embodiments but rather should be  
viewed as limited only by the spirit and scope of the  
appended claims.

A program listing follows in the attached APPENDIX.

- 12 -

```

Agent.java_1      Tue Nov 12 08:01:22 1996      1

1  /* $Header: /com/meitca/hsl/zonesjagents/shared/Agent.java 13 9/30/96 6:23p Walsh $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * Base class for mobile Agent
11 *
12 * $Log: /com/meitca/hsl/zonesjagents/shared/Agent.java $
13 *
14 * 13 9/30/96 6:23p Walsh
15 * clean up javadoc comments
16 *
17 * 12 9/29/96 4:09p Walsh
18 * Give Agent access to AgentPackage information
19 *
20 * 11 9/28/96 6:57p Walsh
21 * Comment out debug println
22 *
23 * 10 9/09/95 4:53p Walsh
24 * Add prepareForTransport, completedTransport methods
25 *
26 * 9 9/04/95 3:18p Noemi
27 * Added static initializer. Moved functionality related to collaboration
28 * to new CollaboratorAgent subclass.
29 *
30 * 8 8/30/96 4:30p Noemi
31 * Changed constructors and finalizer to operate on AgentGroup references
32 * (rather than AgentGroupImpl references).
33 *
34 * 7 8/28/96 2:45p Noemi
35 * Removed SourceSafe conflict garbage.
36 *
37 * 6 8/28/96 2:20p Noemi
38 * Added support for AgentGroups.
39 *
40 * 5 8/23/96 3:52p Walsh
41 * Remove live method (superceded by ad-hoc invocation)
42 *
43 * 4 8/22/96 6:36p Walsh
44 * Remove parameter to live method
45 *
46 * 3 8/13/96 2:23p Walsh
47 * Add some comments & documentation

```

- 13 -

```

Agent.java_1   Tue Nov 12 08:01:22 1996  2
48      *     2    8/09/96 4:49p Walsh
49      * 2
50      */
51 package com.meitca.hsl.zonesagents.shared;
52
53 import java.util.*;
54 import java.net.InetAddress;
55
56 import com.meitca.hsl.zonesagents.conduit.*;
57
58 /**
59  * A base class representing a traveling agent
60  * agents should be derived from the class Agent
61  * travels from machine to machine as dictated by
62  * a specified method will be invoked. After
63  * will be sent to the next host on the itinerary.
64  * programmer should derive a class from Agent
65  * @see ConduitServer
66  * @see AgentGroup
67  * @author Noemi Peclirek
68  * @author Thomas Walsh
69  */
70
71 public abstract class Agent {
72
73     /**
74      * A counter used to generate the Agent
75      * protected static int agentNum = 0;
76
77      * A unique ID representing the host
78      * protected static String host;
79
80      * Instance variables
81      * A unique ID representing the Agent
82      * protected String agentID;
83
84      // A reference to the Agent's AgentPackage
85      // valid when the Agent is actually taken
86      // will be a null reference (for example
87      // when the Agents constructor is executed
88      private AgentPackage itsPackage;
89
90      // Static initializer
91      static {
92
93          /*
94          * Try to obtain the host's name and
95          * construct a pseudo-random name

```

- 14 -

```

Agent.java_1      Tue Nov 12 08:01:22 1996      3

95      * this should never happen.)
96      */
97      try {
98          host = InetAddress.getLocalHost().toString();
99      } catch (Exception e) {
100          host = "__F0R3ID__" + new Random().nextLong();
101      }
102  }
103
104 // Constructors
105 /** Constructs an Agent */
106 public Agent() {
107     /*
108      * Assign a unique agent ID.
109     */
110    try {
111        agentID = host + "/agent" + nextAgentNum();
112        //System.out.println("Agent ID = " + agentID);
113    } catch (Exception e) {
114        System.out.println("Agent constructor error: " + e.getMessage());
115        e.printStackTrace();
116    }
117
118    itsPackage = null;
119
120 }
121
122 // Class methods
123 private final synchronized int nextAgentNum() {
124     return ++agentNum;
125 }
126
127 // Instance methods
128 /**
129  * Retrieves the Agent's ID. An AgentID is a unique String
130  * which identifies a particular instance of an Agent.
131  * @return The Agent's ID.
132 */
133 public final String getAgentID() {
134     return agentID;
135 }
136
137 /**
138  * Converts an Agent to a String. Implementation simply
139  * returns the AgentID.
140  * @return A String representation of the Agent.
141 */

```

- 15 -

```

Agent.java_1           Tue Nov 12 08:01:22 1996   4

142     public final String toString() {
143         return getAgentID();
144     }
145
146     /**
147      * This is called by the ConduitServer immediately prior to
148      * the agent being transported to its next destination. Derived
149      * Agents can override this method to perform any final processing
150      * needed before transport
151      */
152     public void prepareForTransport() {
153
154
155     /**
156      * This is called by the ConduitServer when the Agent arrives at
157      * its new destination. Derived Agents can override this method
158      * to perform any processing needed on arrival
159      */
160     public void completedTransport() {
161
162
163     /**
164      * Returns a reference to the Agents Itinerary. The Itinerary
165      * is only available when the Agent is traveling. It is not
166      * available during the Agent's constructor. If the Itinerary
167      * is not available, this method returns null.
168      * @return The Agents Itinerary or null if the itinerary is
169      *        not available
170      */
171     public Itinerary getItinerary() {
172         if (itsPackage != null) {
173             return itsPackage.getItinerary();
174         } else {
175             return null;
176         }
177     }
178
179     /**
180      * Returns a URL pointing to the Agent's codebase on its home machine.
181      * This URL is used in conjunction with the AgentCodebase class to
182      * retrieve the bytecodes of the Agent and its related classes.
183      * The URL available when the Agent is traveling. It is not
184      * available during the Agent's constructor. If the URL
185      * is not available, this method returns null.
186      * @return The URL of the Agents codebase or null if the URL is
187      *        not available
188      */

```

- 16 -

Agent.java\_1        Tue Nov 12 08:01:22 1996        5

```
189        public String getHomeCodebaseURL() {
190            if (itsPackage != null) {
191                return itsPackage.getHomeCodebaseURL();
192            } else {
193                return null;
194            }
195        }
196        /**
197        * This Method is called internally by the ConduitServer to
198        * provide the Agents with a reference to its package.
199        */
200        public void setPackage(AgentPackage agentPackage) {
201            itsPackage = agentPackage;
202        }
203        }
204        }
205 }
```

- 17 -

```

AgentCodebase.java_1      Tue Nov 12 08:01:22 1996      1
1  /*
2   * $Header: /cork/meitca/hsl/zonesagents/shared/AgentCodebase.java 3
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   */
9   * DESCRIPTION
10  * Class providing accessibility to the home codebase of the agent
11  */
12  */
13  */
14  * 3 11/11/96 5:25p BillF
15  * Made correction to company name
16  */
17  * 2 9/30/96 6:23p Walsh
18  * clean up javadoc comments
19  */
20  * 1 9/28/96 6:35p Walsh
21  * initial version.
22  */
23  */
24 package com.meitca.hsl.zonesagents.shared;
25 import java.net.*;
26 import java.io.*;
27 import sun.net.www.MeteredStream;
28 import com.meitca.hsl.zonesagents.remoteicader.*;
29
30 /**
31 */
32 /**
33  * The AgentCodebase class provides accessibility to the "home
34  * codebase" of an agent. As an agent travels, it may construct
35  * an object whose bytecodes may not be installed on the machine
36  * to which it has travelled. In some cases, the agent may have
37  * already constructed another instance of the same class or, the
38  * user or programmer may have specified that this class was a
39  * "related class" when launching the agent. In both of these
40  * cases, the bytecodes for the class should have already been
41  * loaded into the Agent's MobileCodebase. If the bytecodes for
42  * the new object HAVE NOT already been loaded into the MobileCodebase,
43  * the ConduitServer needs a mechanism for finding an retrieving
44  * these bytecodes. The AgentCodebase provides an mechanism through
45  * which Agents can retrieve code from their home locations.
46  * @see Agent
47  * @see MobileCodebase

```

```

AgentCodebase.java_1      Tue Nov 12 08:01:22 1996    2

48 * @author      Thomas Walsh
49 */
50 public class AgentCodebase {
51 /**
52 * these constants identify the protocols that
53 * can be used to retrieve bytecodes from an
54 * agents codebase.
55 */
56 static final int UNKNOWN = -1;
57 static final int RMI = 0;
58 static final int HTTP = 1;
59 static final int FILE = 2;
60
61 /**
62 * The protocol to use to retrieve bytecodes from the
63 * AgentCodebase
64 */
65
66 /**
67 * An URL pointing back to the AgentCodebase */
68 String itsURL;
69
70 /**
71 * A ClassFileLoader object which may be used to
72 * retrieve agent bytecodes from the local filesystem
73 * (when a file: uri is used to identify the codebase
74 */
75 ClassFileLoader itsClassfileLoader;
76
77 /**
78 * Constructs an AgentCodebase from the given URL.
79 * @param url An URL pointing to the AgentCodebase. This
80 * URL can be:  

81 * <ul>
82 * <li>An HTTP url pointing to a location on a web server
83 * (such as <b>http://host/directory</b>)
84 * <li>An RMI url pointing to a RemoteClassLoader
85 * server (such as <b>rmi://host/RemoteClassLoader/directory</b>)
86 * <li>A FILE url pointing to a local directory
87 * (such as <b>file:C:\directory</b>)
88 * <li>An empty string or null which indicates that the agents code
89 * is on the CLASSPATH of this machine.
90 */
91 * @exception MalformedURLException If the url passed in was invalid
92 */
93 public AgentCodebase(String url) throws MalformedURLException {
94     itsProtocol = -1;

```

```

AgentCodebase.java_1      Tue Nov 12 08:01:22 1996      3

95
96     itsURL = null;
97     itsClassFileLoader= null;
98     parseURL(url);
99
100    if (itsProtocol == FILE)
101        itsClassFileLoader = new ClassFileLoader();
102
103    /**
104     * Retrieve the bytecodes for the given class
105     * @param      classname The name of the class whose bytecodes to
106     *             retrieve
107     * @return     The classes bytecodes in the form of a byte array
108     * @exception  ClassNotFoundException If the class could not be loaded
109     */
110    public byte[] retrieveCode(String classname) throws ClassNotFoundException {
111        switch (itsProtocol) {
112            case RMI:
113                return RemoteClassLoaderImpl.retrieveRemoteClass(itsURL, classname);
114            case HTTP:
115                return retrieveHTTPClass(classname);
116            case FILE:
117                return itsClassFileLoader.loadClassFileFromDirectory(itsURL, classname);
118            default:
119                throw new ClassNotFoundException("Unsupported protocol");
120        }
121    }
122
123    /* Used internally to parse URL passes into ctor */
124    private void parseURL(String url) throws MalformedURLException {
125        if (url != null) {
126            int sep = url.indexOf(':');
127
128            if (sep != -1) {
129                String protocol = url.substring(0, sep);
130                if (protocol.equals("rmi")) {
131                    itsProtocol = RMI;
132                    itsURL = url;
133                } else if (protocol.equals("http")) {
134                    itsProtocol = HTTP;
135                    itsURL = url;
136                    if (itsURL.endsWith("/"))
137                        itsURL = itsURL.substring(0, itsURL.length() - 1);
138                } else if (protocol.equals("file")) {
139                    itsProtocol = FILE;
140                }
141            }
142        }
143    }

```

- 20 -

```

AgentCodebase.java_1      Tue Nov 12 08:01:22 1996    4

142          // strip off the file: portion so all we are left with is
143          // the directory spec of the codebase.
144          itsURL = url.substring(sep+1, url.length());
145
146      } else {
147          // we will assume file since the user might be trying to specify a
148          // DOS style filename (ie C:\WINNT)
149          itsProtocol = FILE;
150          itsURL = url;
151
152      } else {
153          // assume file
154          itsProtocol = FILE;
155          itsURL = url;
156
157      } else {
158          // the URL was null. This means that the Agents code is located somewhere on
159          // the class path of this machine. We will set the protocol to FILE and leave
160          // the url as null
161          itsProtocol = FILE;
162
163      }
164
165      /* Used internally to retrieve .class files from a web server */
166      private byte[] retrieveHTTPClass(String classname) throws ClassNotFoundException {
167          try {
168              // first we will try to load the agent using its fully qualified class
169              // name. If the HTTP URL was http://host/Agents and the class name
170              // was Test.Agent, we will build an URL like the following:
171              // http://host/Agents/test/TestAgent.class
172              URL
173              urlString = new URL(itsURL + "/" + classname.replace('.', '/') + ".class");
174
175              MeteredStream stream = (MeteredStream)classURL.getContent();
176              int length = stream.available();
177              byte data[] = new byte[length];
178
179              stream.read(data);
180              return data;
181          } catch (IOException ierror) {
182              // The fully qualified class name didn't work, so lets try to
183              // strip off all of the package names and search for just the
184              // class name. If the HTTP URL was http://host/Agents and the class name
185              // was Test.Agent, we will build an URL like the following:
186              // http://host/Agents/TestAgent.class
187              int lastSep = classname.lastIndexOf('.');
188              if (lastSep != -1) {
189

```

- 21 -

```
AgentCodebase.java_1      Tue Nov 12 08:01:22 1996      5
+ 1, classname.length() + ".class");
189
URL classURL = new URL(itsURL + "/" + classname.substring(1));
190
MeteredStream stream = (MeteredStream)classURL.getContent();
191
int length = stream.available();
192
byte[] data[] = new byte[length];
193
194
stream.read(data);
195
return data;
196
}
197
catch (Exception error) {
198
    throw new ClassNotFoundException("Could not load class " + classname + " from "
+ itsURL);
199
}
200
else {
201
    throw new ClassNotFoundException("Could not load class " + classname + " from " + itsURL);
202
}
203
}
204
catch (Exception error) {
205
    throw new ClassNotFoundException("Could not load class " + classname + " from " + itsURL);
206
}
207 }
```

- 22 -

```

AgentConstants.java_1      Tue Nov 12 08:01:23 1996      1
1  /*
2   * $Header: /com/meitca/hsl/zonesjagents/shared/AgentConstants.java 2      11/11/96 5:25p Billip $
3   *
4   * Copyright 1996 Horizon Systems Laboratory,
5   * Mitsubishi Electric Information Technology Center America.
6   * All rights reserved.
7   *
8   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
9   *
10  * DESCRIPTION
11  * Constants used by Java Agents system.
12  * $Log: /com/meitca/hsl/zonesjagents/shared/AgentConstants.java $
13  *
14  * 2      11/11/96 5:25p Billip
15  *       Made correction to company name
16  *
17  * 1      10/10/96 5:50p Walsh
18  *       initial version.
19  *
20  */
21 package com.meitca.hsl.zonesjagents.shared;
22
23 import java.io.File;
24
25 /**
26  * Class contains constants used within the Java Agent System.
27  */
28 public class AgentConstants {
29 /**
30  * Contains the value of the "zones.home" property (which
31  * points to the Zones installation directory)
32  */
33 public static String ZONES_HOME;
34
35 /**
36  * Contains the value of the "zones.user.home" property (which
37  * points to the Zones user specific directory)
38  */
39 public static String ZONES_USER_HOME;
40
41 /**
42  * Contains the path of the images subdirectory of the Zones
43  * installation directory.
44  */
45 public static String IMAGE_DIR;
46
47 */

```

- 23 -

```

AgentConstants.java_1      Tue Nov 12 09:01:23 1996   2

48      * Contains the path of the Agent subdirectory of the Zones
49          * installation directory.
50      */
51      public static String      AGENT_DIR;
52
53      /**
54          * Contains the path of the lib subdirectory of the Zones/Agents
55          * installation directory.
56      */
57      public static String      AGENT_LIB_DIR;
58
59      /**
60          * Contains the path of the Agent subdirectory of the Zones
61          * installation directory.
62      */
63      public static String      USER_AGENT_DIR;
64
65      /**
66          * Static Initializer is used to initialize some of the CONSTANTS *
67      static {
68          ZONES_HOME = System.getProperty("zones.home", File.separator + "Zones");
69          if (ZONES_HOME != null) {
70              // terminate the directory name with a separator if
71              // it's not already terminated
72              if (!ZONES_HOME.endsWith(File.separator))
73                  ZONES_HOME += File.separator;
74
75          IMAGE_DIR = ZONES_HOME + "images" + File.separator;
76          AGENT_DIR = ZONES_HOME + "Agents" + File.separator;
77          AGENT_LIB_DIR = AGENT_DIR + "lib" + File.separator;
78      }
79
80          ZONES_USER_HOME = System.getProperty("zones.user.home");
81          if (ZONES_USER_HOME == null) {
82              // the zones user.home Property was not defined.
83              // try the user.home Property
84              ZONES_USER_HOME = System.getProperty("user.home", "");
85              if (ZONES_USER_HOME != null) {
86                  // terminate the directory name with a separator if
87                  // it's not already terminated
88                  if (!ZONES_USER_HOME.endsWith(File.separator))
89                      ZONES_USER_HOME += File.separator;
90
91          ZONES_USER_HOME += ".Zones";
92      }
93
94

```

- 24 -

Agentconstants.java\_1        Tue Nov 12 08:01:23 1996        3

```
95        if (ZONES_USER_HOME != null) {
96            // terminate the directory name with a separator if
97            // it's not already terminated
98            if (!ZONES_USER_HOME.endsWith(File.separator))
99                ZONES_USER_HOME += File.separator;
100
101            USER_AGENT_DIR = ZONES_USER_HOME + "Agents" + File.separator;
102
103        }
104 }
```

- 25 -

```

ClassFileLoader.java_1      Tue Nov 12 08:01:23 1996      1
1  /* $Header: /com/meitca/hsl/zonesagents/shared/ClassFileLoader.java 3      11/11/96 5:25p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory.
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  *
9  * DESCRIPTION
10 * Loads a ".class" file off of the local filesystem.
11 *
12 * $Log: /com/meitca/hsl/zonesagents/shared/ClassFileLoader.java $
13 *
14 * 3 11/11/96 5:25p Billp
15 *     Made correction to company name
16 *
17 * 2 9/30/96 6:23p Walsh
18 *     clean up javadoc comments
19 *
20 * 1 9/28/96 6:35p Walsh
21 *     initial version
22 *
23 */
24 package com.meitca.hsl.zonesagents.shared;
25
26 import sun.tools.java.*;
27 import java.io.*;
28
29 /**
30 * The ClassFileLoader class loads a Classes bytecodes off of the
31 * local filesystem. The user can either specify a particular directory
32 * from which the bytecodes should be retrieved, have the ClassFileLoader
33 * search the system CLASSPATH or specify a custom Class Path which should
34 * be searched for the bytecodes.
35 * @author Thomas Walsh
36 */
37 public class ClassFileLoader {
38 /**
39 * The ClassPath which should be searched for ClassFiles. This could
40 * represent the system CLASSPATH, or a custom ClassPath specified by the
41 * user.
42 */
43 ClassPath itsClassPath;
44
45 /**
46 * Constructs a ClassFileLoader object. This constructor sets the
47 * ClassFileLoader up so that it will search the system's CLASSPATH.

```

```

ClassFileLoader.java_1   Tue Nov 12 08:01:23 1996    2

48      */
49      public ClassFileLoader() {
50          itsClassPath = new ClassPath(System.getProperty("java.class.path"));
51      }
52
53      /**
54      * Constructs a ClassFileLoader object. The pathstr parameter specifies
55      * a custom ClassPath from which bytecodes are to be retrieved.
56      * @param pathstr A custom ClassPath from which bytecodes are to be
57      * retrieved
58      */
59      public ClassFileLoader(String pathstr) {
60          itsClassPath = new ClassPath(pathstr);
61      }
62
63      /**
64      * Loads a class from the objects class path. Depending on which
65      * version of the constructor was used, this could mean that the
66      * system's CLASSPATH is searched or that a custom user-defined
67      * ClassPath is searched.
68      * @param classname The fully qualified class name of the class to
69      * be loaded.
70      * @return the bytecodes for the requested class in the form of a
71      * byte array
72      * @exception ClassNotFoundException If the class could not be located on
73      * the ClassPath
74      */
75      public byte[] loadClassFileFromClassPath(String classname) throws ClassNotFoundException {
76          String filename = classname.replace('.', File.separatorChar) + ".class";
77          ClassFile file = itsClassPath.getFile(filename);
78
79          if (file != null) {
80              try {
81                  long length = file.length();
82                  InputStream stream = file.getInputStream();
83                  byte data[] = new byte[(int)length];
84
85                  stream.read(data);
86
87                  return data;
88              } catch (IOException e) {
89                  throw new ClassNotFoundException(e.getMessage());
90              }
91          }
92          throw new ClassNotFoundException("Classfile not found on ClassPath: " + classname);
93      }
94

```

-27-

```

JASProperties.java_1      Tue Nov 12 08:01:23 1996      1
1  /*
2   *      $Header: /com/meitca/hsl/zones\agents\shared/JASProperties.java 4      11/11/96 5:25p Billip $
3   *
4   *      Copyright 1996 Horizon Systems Laboratory,
5   *      Mitsubishi Electric Information Technology Center America
6   *      All rights reserved.
7   *
8   *      CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
9   */
10  /*
11   *      Zones Agent Property File implementation.
12   *      $Log: /com/meitca/hsl/zones\agents\shared/JASProperties.java $
13   *          4      11/11/96 5:25p Billip
14   *          Made correction to company name
15   *          3      10/10/96 5:56p Walsh
16   *          Make use of constants in AgentConstants class
17   *          2      10/10/96 5:56p Walsh
18   *          Add default constructor
19   *          1      10/07/96 2:30p Walsh
20   *          initial versions
21   *
22   */
23  /*
24   */
25  /*
26 package com.meitca.hsl.zones\agents\shared;
27
28 import java.io.*;
29 import javax.swing.*;
30
31 /**
32 *      The JASProperties supports the storage of
33 *      user specific and server specific properties.
34 *      <ul>
35 *          <li>User specific properties are stored in
36 *              zones\user\home\agents directory. If the
37 *              zones\user\home\System property is not defined,
38 *              user properties are stored in the
39 *              user\home\Zones\Agents directory
40 *          <li>Server specific properties are stored in
41 *              zones\home\agents\lib
42 *      </ul>
43 *      @see
44 *          Author      Joe DiCicco
45 *          Author      Tom Walsh
46 */
47 public class JASProperties extends PropertyFile {

```

- 28 -

classFileLoader.java\_1        Tue Nov 12 08:01:23 1996        4

```
141              ) else {        return data;
142              ) else {
143              throw new ClassNotFoundException("Could not find " + classname +
144              " in directory " + dir);
145              )
146              }
147              ) catch (IOException e) {
148              throw new ClassNotFoundException("Could not load " + classname +
149              " Error: " + e.getMessage());
150              )
151              )
152              }
```

- 29 -

```

 95  /**
 96   * Loads a class from a particular directory.
 97   * @param dir The directory from which to retrieve the class. If
 98   *           this parameter is null or an empty string, the ClassPath
 99   *           is searched for the class
100  *           @param classname The fully qualified class name of the class to
101  *           be loaded.
102  *           @returns The bytecodes for the requested class in the form of a
103  *           byte array
104  *           @exception ClassNotFoundException If the class could not be located on
105  *           the ClassPath
106  */
107 public byte[] loadClassFromFileFromDirectory(String dir, String classname) throws ClassNotFoundException {
108   // if a null or an empty string was passed in for dir, fall back to
109   // searching the class path for the class
110   if ((dir == null) || (dir.length() == 0))
111     return loadClassFromFileFromClassPath(classname);
112
113   // Place a separator char on the end of the directory specification
114   // if it does not already have one
115   if (!dir.endsWith(File.separator)) {
116     dir = dir + File.separator;
117   }
118
119   String fulifilename = dir + classname.replace('.', File.separatorChar) + ".class";
120
121   try {
122     File file = new File(fullfilename);
123
124     if (file.exists()) {
125       long length = file.length();
126       FileInputStream stream = new FileInputStream(file);
127       byte data[] = new byte[(int)length];
128
129       stream.read(data);
130
131     } else {
132       int lastSep = classname.lastIndexOf('.');
133       if (lastSep != -1) {
134         String filename = dir + classname.substring(lastSep + 1, classname.length());
135
136         file = new File(filename);
137         length = file.length();
138         FileInputStream stream = new FileInputStream(file);
139         byte data[] = new byte[(int)length];
140
141         stream.read(data);
142
143       }
144     }
145   } catch (Exception e) {
146     System.out.println("Error loading class file " + fullfilename);
147     e.printStackTrace();
148   }
149 }

```

## JASProperties.java\_1

Tue Nov 12 08:01:23 1996 2

```

48
49      /**
50       * This constant should be passed into constructor to
51       * specify that this object will contain <b>User</b>
52       * specific properties.
53     */
54     public static final int SERVER = 0;
55
56   /**
57    * This constant should be passed into constructor to
58    * specify that this object will contain <b>User</b>
59    * specific properties.
60   */
61   public static final int USER = 1;
62
63   private static final String PROP_FILE_SUFFIX = ".Properties";
64   private static final String DEFAULT_DESC = "Zones Property File";
65
66   /**
67    * Opens a Properties file and constructs a JASProperties object.
68    * @param
69    *   appName A unique name used to identify the application.
70    *   *
71    *   The <i>appName</i> is used in constructing the name of the
72    *   properties file. The properties file will have a name of the form
73    *   *
74    *   "appName.properties"
75    *   *
76    *   @param
77    *   *
78    *   type Identifies the type of properties file to open.
79    *   *
80    *   zones.user.home System property is not defined,
81    *   zones.user.home Properties.SERVER</b>. The <i>type</i> parameter
82    *   controls where the properties file is opened from.
83    *   *
84    *   <ul>
85    *   *   *
86    *   *   *   *
87    *   *   *   *   *
88    *   *   *   *   *   *
89    *   *   *   *   *   *   *
90    *   *   *   *   *   *   *   *
91    *   *   *   *   *   *   *   *   *
92    *   *   *   *   *   *   *   *   *   *
93    *   *   *   *   *   *   *   *   *   *   *
94    *   *   *   *   *   *   *   *   *   *   *   *

```

- 30 -

JASProperties file could not

- 3 -

```

JASProperties.java_1      Tue Nov 12 08:01:23 1996      3

95      * @exception IOException If an error occurred while attempting to
96      *          access the Properties file
97      * @exception IllegalArgumentException If the value passes into the
98      *          type argument was not valid.
99      */
100     public JASProperties(String appName, int type, String description,
101                           boolean createIfNecessary) throws FileNotFoundException,
102                                         IOException, IllegalArgumentException {
103         itsDescription = (description != null) ? description : DEFAULT_DESC;
104         // Open the default JAS security properties file.
105         itsFile = locatePropertiesFile(appName, type, createIfNecessary);
106         loadProperties();
107     }
108
109
110
111
112 /**
113  * Builds an "empty" Properties object. This method can be used
114  * to construct a Properties object which returns only default values.
115  * This behavior may be desired if the Properties file can not be found
116  * and the program wishes to continue execution using its default values
117  * for preferences.
118  */
119 public JASProperties() {
120
121
122
123
124 /**
125  * Builds a Properties filename based on the appName and properties
126  * file type
127  */
128 private File locatePropertiesFile(String appName, int type,
129                                   boolean createIfNecessary)
130         throws IllegalArgumentException, FileNotFoundException {
131
132     String propDirName;
133     if (type == SERVER) {
134         // We are accessing a server properties file.
135         // Server specific properties are stored in
136         // the directory zones.home\Agents\Lib.
137
138         propDirName = AgentConstants.AGENT_LIB_DIR;
139     } else if (type == USER) {
140         propDirName = AgentConstants.USER_AGENT_DIR;
141     } else {

```

```

JASProperties.java_1

        throw new IllegalArgumentException("Bad Properties File Type: " + type);
    }

    // See if the properties directory exists
    File propDir = new File(propDirName);
    if (!propDir.exists()) {
        // The directory does not exist.
        // See if we should create it.
        if (createIfNecessary) {
            if (!propDir.mkdirs()) {
                // We could not create the directory.
                // Throw an IOException
                throw new IOException("Could not create properties directory " + propDir.toString());
            }
        } else {
            // The directory did not exists.
            // Throw a FileNotFoundException
            throw new FileNotFoundException("Properties directory " + propDir.toString() + " did not exists");
        }
    }

    // see if the properties file exists
    File propFile = new File(propDirName + File.separator + appName + PROP_FILE_SUFFIX);

    if (!propFile.exists()) {
        // The directory does not exist.
        // See if we should create it.
        if (createIfNecessary) {
            // create the file.
            FileOutputStream out = new FileOutputStream(propFile);
            PrintStream print = new PrintStream(out);
            print.println("#" + itsDescription);
            print.close();
            // That should have created the file
        } else {
            // The properties file did not exists.
            // Throw a FileNotFoundException
            throw new FileNotFoundException("Properties file " + propFile.toString() + " did not exists");
        }
    }
    return propFile;
}

```

- 33 -

JASProperties.java\_1 Tue Nov 12 08:01:23 1996 5

```

PropertyFile.java_1      Tue Nov 12 08:01:23 1996      1
1  /*
2   * $Header: /ccm/meitca/ns1/zonesagents/shared/PropertyFile.java 2      11/11/96 5:25p BillP $
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   * DESCRIPTION
10  * Properties (Preference) File object.
11  *
12  * $Log: /com/meitca/ns1/zonesagents/shared/PropertyFile.java $
13  * 2      11/11/96 5:25p BillP
14  *      * Made correction to company name
15  *      * 10/07/96 5:07p Walsh
16  *      * initial versions
17  *      * 1
18  *      *
19  *      *
20  */
21 package com.meitca.ns1.zonesagents.shared;
22
23 import java.io.*;
24 import java.util.*;
25
26 /**
27  * The PropertyFile object extends the behavior of the Properties class
28  * by associating the Properties with a disk file. The PropertyFile
29  * object automatically loads the Properties out of the file (using the
30  * Properties.load method) and supplies a method for saving of any updated
31  * Properties back to disk, called saveProperties.
32  * @see Properties
33  * @author Jce Dicelie
34  * @author Tom Walsh
35  */
36 public class PropertyFile extends Properties {
37
38  /** The name of the properties file corresponding to this Properties object */
39  protected File itsFile;
40
41  /** A description of this Properties object. */
42  protected String itsDescription;
43
44  /**
45  * Opens the specified Properties file and constructs a
46  * JASProperties object.
47  */

```

- 35 -

```

PropertyFile.java_1      Tue Nov 12 08:01:23 1996      2

48      * @param
49          * @param
50          *
51          *
52          * @exception  FileNotFoundException  If the Properties file could not
53          *             be found and the createIfNecessary parameter is false.
54          *
55          * @exception  IOException  If an error occurred while attempting to
56          *             access the Properties file
57
58      public PropertyFile(String propertiesFileName, String description)
59          throws FileNotFoundException, IOException {
60
61          itsFile = new File(propertiesFileName);
62          itsDescription = description;
63          loadProperties();
64
65          /**
66          * Protected constructor that can be used by derived classes.
67          */
68          protected PropertyFile() {
69              itsFile = null;
70              itsDescription = null;
71              // The actual work of loading properties will be
72              // done by derived class
73          }
74
75          // Loads Properties from the Properties File
76          protected void loadProperties()
77              throws FileNotFoundException, IOException {
78
79              // open an input stream to the file specified
80              FileInputStream in = new FileInputStream(itsFile);
81
82              // load the properties
83              load(in);
84
85          }
86
87          /**
88          * Saves the Properties to the corresponding Properties File
89          * @exception  FileNotFoundException  if the Properties file could not be
90          *             located
91          * @exception  IOException  if an error occurs while accessing the
92          *             properties file
93
94      public void saveProperties()

```

- 36 -

```
PropertyFile.java_1      Tue Nov 12 08:01:23 1996      3

95          throws FileNotFoundException, IOException {
96
97          // open an input stream to the file specified
98          FileInputStream in = new FileInputStream(itsFile);
99
100         // load the properties
101         save(out, itsDescription);
102     }
103
104     /**
105      * Refreshes the Properties object by re-reading the Properties file.
106      * Changes made to the Properties object will be lost.
107      * IOException FileNotFoundExeption if the Properties file could not be
108      * located
109      * IOException if an error occurs while accessing the
110      * Properties file
111     */
112     public void refreshProperties()
113         throws FileNotFoundException, IOException {
114         loadProperties();
115     }
116
117
118     }
119
120
121
122
```

```

1  /* $Header: /ccm/mejice/hsl/zonesagents/bootstrap/AgentClassInfoGatherer.java 5   11/11/96 5:13p Billip $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * AgentClassInfoGatherer class peek's into an agents .class file to
11 * retrieve information needed for agent launching.
12 *
13 * SLUG: com.mitscience.zonesagents.bootstrap/AgentClassInfoGatherer.java $
14 * 5 11/11/96 5:13p Billip
15 * Name correction to company name.
16 * 5/30/96 5:13p Walsh
17 * fix up javadoc comments
18 * 4 9/27/96 6:10% Dicelie
19 * minor change to lcp which verified that the class to be launched was
20 * subclasses from Agent.
21 * 3 9/27/96 6:10% Dicelie
22 * minor change to lcp which verified that the class to be launched was
23 * subclasses from Agent.
24 * 2 9/10/96 11:46a Walsh
25 * remove bogus imports that javac considers illegal
26 * 1 9/29/96 5:23p Walsh
27 * initial version.
28 *
29 *
30 */
31 import com.mitscience.zonesagents.comms.AgentSkeleton;
32 package com.mitscience.zonesagents.bootstrap;
33
34 import java.io.*;
35 import com.mitscience.zonesagents.comms.AgentSkeleton;
36
37 /**
38 * This object peek's into the .class file of an agent to retrieve
39 * some information needed for launching the agent.
40 * AgentClassInfoGatherer verifies that the .class file indicated
41 * does contain a class derived from Agent and then it verifies the
42 * existence of the agent's invocation skeleton and finally loads
43 * the skeleton class into memory and retrieves a list of the
44 * agents methods from the skeleton.
45 * @see Agent
46 * @see Bootstrap
47 * @see AgentLaunchWizard

```

```

AgentClassInfoGatherer.java_1 Tue Nov 12 07:44:46 1996

48 * @author Thomas Walsh
49 */
50 public class AgentClassInfoGatherer {
51     /** The names of the agents methods */
52     String[] itsAgentsMethods;
53
54     /**
55      * Constructs a AgentClassInfoGatherer object. Loads
56      * and its skeleton. Retrieves a list of the agent's
57      * methods. agentDir The directory containing the
58      * agent's class file.
59      * @param agentName The name of the
60      * @throws FileNotFoundException Thrown if the
61      * agentDir could not be opened.
62      * @throws IOException If an I/O error occurs while
63      * access the agent's class
64      * @throws ClassFormatException If the agent's class
65      * file contains a valid
66      * @throws NotAgentException If the agent's class
67      * file does not contain an agent
68      * @throws ClassNotFoundException If an error occurs
69      * trying to access the
70      * @throws NoAgentSkeletonException If the agent's
71      * skeleton could not be
72      */
73     public AgentClassInfoGatherer(String agentDir, String
74         throws FileNotFoundException, IOException,
75         NotAgentException, ClassFormatException,
76         NoAgentSkeletonException);
77
78     /**
79      * Let the loadAgentClassInfo(agentDir, agentFilename) :
80
81
82      * Verifies the existence and correctness of the agent
83      * and the agent's skeleton. Retrieves a list of the
84      * agent's methods internally by the constructor
85      * @param agentName The name of the
86      * @throws FileNotFoundException Thrown if the
87      * agentDir did not exist.
88      * @throws IOException If an I/O error occurs while
89      * access the agent's class
90      * @throws ClassFormatException If the agent's class
91      * file contains a valid
92      * @throws NotAgentException If the agent's class
93      * file does not contain an agent
94

```

- 39 -

```

AgentClassInfoCatherer.java_1      Tue Nov 12 07:44:46 1996      3

95      * @exception NotAgentException if the agent's .class file does
96      *          not contain an agent.
97      * @exception ClassNotFoundException if an error occurred while
98      *          trying to access the agents .class file.
99      * @exception NotAgentSkeletonException if the agent's invocation
100     *          skeleton could not be found.
101
102  private void loadAgentClassInfo(String agentDir, String agentFilename)
103      throws FileNotFoundException, IOException, ClassFormatError,
104          NotAgentException, ClassNotFoundException,
105          NotAgentSkeletonException {
106
107      // Load the Agent class
108      BootStrapClassLoader classLoader = new BootStrapClassLoader(agentDir);
109      Class agent = classLoader.loadAgentClassFromFile(agentFilename, true);
110
111      // Verify that it is actually an Agent
112      Class superClass = agent.getSuperclass();
113
114      // STABILITY: Any way to get these hard coded strings out of here?
115      String superClassName;
116
117      superClass = superClass.getName();
118      if (superClassName.equals("java.lang.Object")) {
119          throw new NotAgentException(agentFilename + " does not contain an agent");
120      }
121
122      superClass = superClass.getSuperclass();
123
124      while (!superClassName.equals("com.melca.hsl.zonesagents.skel.Agent")) {
125
126          // Ok, we have an agent...
127          // Let's try to load the skeleton
128
129          try {
130              // First lets strip the ".class" from the filename
131              // STABILITY: Any way to get these hard coded strings out of here?
132              String skelClassName = agentFilename.substring(0,
133                  agentFilename.length() - ".class".length());
134
135              // Append on the _Skel
136              skelClassName = skelClassName + "_Skel.class";
137
138              // now try to load the skeleton class
139              Class skelClass = classLoader.loadAgentClassFromFile(skelClassName, true);
140
141          // Finally retrieve the agents methods.

```

-40-

```
AgentClassInfoGatherer.java_1      Tue Nov 12 07:44:46 1996      4

142     AgentSkeleton skel = (AgentSkeleton)skelClass.newInstance();
143     itsAgentsMethods = skel.getMethods();
144   } catch (Exception e) {
145     throw new ClassNotFoundException("Could not find skeleton for agent " + agentFilename);
146   }
147 }

148 /**
149 * Retrieves a list of the agents methods. This list is used
150 * by the Agent Launch GUI in order to present the user with
151 * a drop down list containing the eligible methods of the agent.
152 */
153 public String[] getAgentsMethods() {
154   return itsAgentsMethods;
155 }
156 }
157 }
```

```

1  /* $Header: /ccm/meitca/hsl/zonesagents/bootstrap/AgentLaunchInformation.java 1
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  *
9  * DESCRIPTION
10 * AgentLaunchInformation encapsulates the information needed to
11 * bootstrap the agent into the system.
12 *
13 * $Log: /com/meitca/hsl/zonesagents/bootstrap/AgentLaunchInformation.java $
14 * Revision 1.1 1996-01-11 13:51:12 Billp
15 * Made correction to company name.
16 * 3 9/12/95 6:35p Welsh
17 * Fix up javadoc comments
18 * 2 9/11/95 4:46p Welsh
19 * add itsAgent member
20 *
21 * 1 6/23/96 3:21p Welsh
22 * initial version.
23 *
24 */
25
26
27
28 package com.meitca.hsl.zonesagents.bootstrap;
29 import com.meitca.hsl.zonesagents.conduit.Itinerary;
30 import com.meitca.hsl.zonesagents.shared.Agent;
31
32
33 /**
34 * The AgentLaunchInformation class encapsulates the information
35 * needed by the Bootstrap class in order to launch and Agent.
36 * This class is used internally by the AgentLaunchWizard and
37 * the Agent command line launch tool.
38 * @see Agent
39 * @see Bootstrap
40 * @see AgentLaunchWizard
41 * @see Thomas Welsh
42 * @author Thomas Welsh
43 */
44 public class AgentLaunchInformation {
45
46     /*
47     * The directory (on the local host) containing the agent's
48     * .class file as well as the .class files of any related

```

- 42 -

```

AgentLaunchInformation.java 1   Tue Nov 12 07:44:46 1996   2

48     * classes.
49     */
50     public String           itsAgentDirectory;
51
52     /** The agent's .class file */
53     public String           itsAgentFile;
54
55     /**
56      * An array containing the filenames of any other classes
57      * that should be sent along with the agent.  The ConduitServer
58      * has the ability to communicate back to the agents home machine
59      * to retrieve classes needed at runtime (if the RemoteLoader server
60      * is running).  With this parameter a programmer can bypass this
61      * and send any needed classes up front.
62
63     public String[]          itsRelatedClasses;
64
65     /** The agents Itinerary */
66     public Itinerary          itsItinerary;
67
68     /**
69      * The agent itself.  This member can needs to be filled in if the
70      * caller wants to call a method on the agent other than the default
71      * constructor.  If this field is left blank, then the
72      * bootstrap launch method will construct the Agent contained in
73      * the file itsAgentDirectory by calling its default constructor.
74      * This method gives the programmer the option of constructing an
75      * agent with a class other than the default.
76
77     public Agent             itsAgent;
78
79     /**
80      * A listing of the agents methods.  This member variable need not
81      * be filled in in order to launch the agent, but is used internally
82      * by the AgentLaunchWizard
83
84     public String[]          itsMethods;
85
86     /**
87      * Constructs an empty AgentLaunchInformation object */
88     public AgentLaunchInformation() {
89         itsAgentDirectory = new String("");
90         itsAgentFile = new String("");
91         itsItinerary = new Itinerary();
92         itsAgent = null;
93     }

```

- 43 -

## AgentLaunchWizard.java\_1

Tue Nov 12 07:44:46 1996

1

```

1  /* $Header: /cvs/meitaca/hsl/zones/jagents/bootstrap/AgentLaunchWizard.java 10
2   * Copyright 1996 Horizon Systems Laboratory,
3   * Mitsubishi Electric Information Technology Center America.
4   * All rights reserved.
5   *
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC I.T.A.
8   *
9   * DESCRIPTION
10  * GUI tool for launching agents into system
11  *
12  * $Log: /cvs/meitaca/hsl/zones/jagents/bootstrap/AgentLaunchWizard.java $
13  *
14  * Revision 1.11 96-5-13p Billp
15  * Made correction to company name.
16  *
17  * Revision 1.10 96-5-16p Billp
18  * Make use of properties in AgentConstants class
19  *
20  * Revision 1.9 96-5-25p Walsh
21  * Fix name collision between bootstrap.StringResources and
22  * util.StringResources
23  *
24  * Revision 1.8 96-4-14p Walsh
25  * Save name & location of last agent launched. Restart Wizard pointing
26  * at same agent
27  *
28  * Revision 1.7 96-3-25p Walsh
29  * Set up code to look in zones.home/images for GIF file
30  *
31  * Revision 1.6 96-3-25p Walsh
32  * fix up jediton comments
33  *
34  * Revision 1.5 96-3-25p Walsh
35  * Print exception to stderr on launchException
36  *
37  * Revision 1.4 96-3-25p Walsh
38  * replace GuiMessageDialog with com.meitaca.hsl.util.InfoDialog. Create
39  * security manager in main
40  *
41  * Revision 1.3 96-12-34p Walsh
42  * Pop up dialog on any problems launching the agent
43  *
44  * Revision 1.2 96-5-21p Walsh
45  * initial version
46  *
47  */

```

-44-

```

AgentLaunchWizard.java_1      Tue Nov 12 07:44:46 1996      2

48  package com.meitca.hsl.zonesjagents.bootstrap;
49
50  import java.awt.*;
51  import java.io.*;
52  import java.util.*;
53
54  import com.meitca.hsl.util.*;
55  import com.meitca.hsl.zonesjagents.security.*;
56  import com.meitca.hsl.zonesjagents.shared.*;
57
58  /**
59   * The AgentLaunchWizard provides a wizard-like GUI for launching
60   * Java Agents into the system
61   * @see Agent
62   * @see BootStrap
63   * @see AgentLaunchInformation
64   * @see Wizard
65   * @author Thomas Walsh
66   */
67  public class AgentLaunchWizard extends Wizard {
68      /** The launch information */
69      AgentLaunchInformation itsAgentInfo;
70
71      /** The filename of the GIF */
72      static final String GIF_FILENAME = AgentConstants.IMAGE_DIR + "AgentLaunchWizard.gif";
73
74      /** Properties (Preference) object */
75      JASProperties itsProperties;
76
77      static final String LAST_AGENT_DIR="zones.agent.bootstrap.LastAgentDir";
78      static final String LAST_AGENT_FILE="zones.agent.bootstrap.LastAgentFile";
79
80      /**
81       * Constructs an AgentLaunchWizard object. Brings up the GUI and
82       * sets up the first panel.
83       */
84      public AgentLaunchWizard() {
85          super(com.meitca.hsl.zonesjagents.bootstrap.StringResources.WIZARD_TITLE,
86                GIF_FILENAME);
87          itsAgentInfo = new AgentLaunchInformation();
88
89      try {
90          itsProperties = new JASProperties(
91              com.meitca.hsl.zonesjagents.bootstrap.StringResources.WIZARD_APPNAME,
92              JASProperties.USER,
93              com.meitca.hsl.zonesjagents.bootstrap.StringResources.PROP_DESC,
94

```

- 45 -

```

AgentLaunchWizard.java_1      Tue Nov 12 07:44:46 1996      3

95
96      ) catch (Exception e) {
97          itsProperties = null;
98      }
99
100     if (itsProperties != null) {
101         itsAgentInfo.itsAgentDirectory = itsProperties.getProperty(LAST_AGENT_DIR, "");
102         itsAgentInfo.itsAgentFile = itsProperties.getProperty(LAST_AGENT_FILE, "");
103     }
104
105     buildPanels();
106     firstPanel();
107
108     resize(570, 370);
109     show();
110
111 /**
112 * Builds the panels of the wizard. The AgentLaunchWizard
113 * contains three panels 1) a Panel for specifying the agent's .class
114 * file (names FilePanel), 2) a panel for specifying the related
115 * class files for sending with the agent (called RelatedFilesPanel) and
116 * 3) a panel for setting up the agents itinerary (called ItineraryPanel)
117 */
118 protected void buildPanels() {
119     FileInputStream panel1 = new FileInputStream(this, itsAgentInfo);
120     RelatedFilesPanel panel2 = new RelatedFilesPanel(this, itsAgentInfo);
121     ItineraryPanel panel3 = new ItineraryPanel(this, itsAgentInfo);
122 }
123
124 /**
125 * The finish method is called by the Wizard base class when the
126 * user has successfully filled all the necessary information
127 * into all the panels and has pressed the "Finish" button. We now
128 * have all of the information we need so we'll try to launch the agent.
129 */
130 protected void finish() {
131     try {
132         /**
133          * launch the agent
134          * BootStrap.launchAgent(itsAgentInfo);
135
136         if (itsProperties != null) {
137             try {
138                 itsProperties.put(LAST_AGENT_DIR, itsAgentInfo.itsAgentDirectory);
139                 itsProperties.put(LAST_AGENT_FILE, itsAgentInfo.itsAgentFile);
140                 itsProperties.saveProperties();
141             } catch (Exception e) {

```

- 46 -

Tue Nov 12 07:44:46 1996 4

```
AgentLaunchWizard.java_1

142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170

    }
}

/*
 * main routine supplied so that AgentLaunchWizard can be initiated
 * from the command line
 */
public static void main(String args[]) {
    /*
     * Create and install the security manager
     */
    System.setSecurityManager(new InsecurityManager());
    AgentLaunchWizard gui = new AgentLaunchWizard();
}
```

```

agentLaunchwizardPanel.java_1 Tue Nov 12 07:44:47 1996 1
1 /* SHeader: /ccm/meitca/its1/zonesjagents/bootstrap/AgentLaunchWizardPanel.java 4 11/11/96 5:13p Billp $ */
2 *
3 * Copyright 1996 Horizon Systems Laboratory.
4 * Mitsubishi Electric Information Technology Center America.
5 * All rights reserved.
6 *
7 * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8 *
9 * DESCRIPTION:
10 * A subclass of WizardPanel for use in the AgentLaunchWizard
11 *
12 * $Log: /com/meitca/hs1/zonesjagents/bootstrap/AgentLaunchWizardPanel.java $
13 *
14 * Revision 1.4 11/11/96 5:13p Billp
15 * Made correction to company name.
16 *
17 * Revision 1.3 9/30/96 6:35c Walsh
18 * fix up javadoc comments
19 *
20 * Revision 1.2 9/10/96 11:46c Walsh
21 * remove bogus imports that javac considers illegal
22 *
23 * Revision 1.1 8/29/96 5:21p Walsh
24 * initial version
25 *
26 */
27 package com.meitca.hsl.zonesjagents.bootstrap;
28
29 import java.awt.*;
30
31 import com.meitca.hsl.util.Wizard;
32 import com.meitca.hsl.util.WizardPanel;
33
34 /*
35 */
36 * AgentLaunchWizardPanel is a base class for the panels of the
37 * AgentLaunchWizard. All of the panels in the launch wizard need
38 * shared access to the AgentLaunchInformation object describing the
39 * parameters of the launch. This base class provides that
40 * access.
41 * @see AgentLaunchWizardPanel
42 * @see Bootstrap
43 * @see AgentLaunchInformation
44 * @see AgentLaunchWizard
45 * @author Thomas Walsh
46 */
47 public class AgentLaunchWizardPanel extends WizardPanel {

```

- 48 -

AgentLaunchWizardPanel.java\_1      Tue Nov 12 07:44:47 1996      2

```
48     /** The launch parameters of the Wizard*/
49     protected AgentLaunchInformation     itsAgentInfo;
50
51     /*
52       * Constructs an AgentLaunchWizardPanel.
53       * @param     wizard     The Wizard owning this panel (needed by super)
54       * @param     info     info The (shared) reference to the agent launch information
55       */
56     public AgentLaunchWizardPanel(Wizard wizard, AgentLaunchInformation info) {
57         super(wizard);
58         itsAgentInfo = info;
59     }
60 }
61
```

- 49 -

```

Bootstrap.java_1      Tue Nov 12 07:44:47 1996      1
1  /* $Header: /com/meitca/hsl/zonesagents/bootstrap/Bootstrap.java 11 11/11/96 5:13p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC IPA.
8  */
9  * DESCRIPTION
10 * Module for boot strapping agents into the system.
11 *
12 * $Log: /com/meitca/hsl/zonesagents/bootstrap/Bootstrap.java $
13 * Revision 1.1 - 11/11/96 5:13p Billp
14 * 11 11/11/96 5:13p Billp
15 * Made correction to company name.
16 *
17 * Revision 1.0 - 10/01/96 10:39a Walsh
18 * fix @exception javadoc tag
19 *
20 * Revision 0.9 - 9/30/96 7:33p Walsh
21 * add more javadoc
22 *
23 * Revision 0.8 - 9/29/96 6:56p Walsh
24 * Add new version of launchAgents
25 *
26 * Revision 0.7 - 9/11/96 4:59p Walsh
27 * allow launchInfo.itsRelatedClasses to be null
28 *
29 * Revision 0.6 - 9/11/96 4:46p Walsh
30 * if caller specified LaunchInfo.itsAgent, use that rather than
31 * constructing a new one.
32 *
33 * Revision 0.5 - 9/10/96 11:46a Walsh
34 * remove bogus imports that javac considers illegal
35 *
36 * Revision 0.4 - 9/09/96 3:46p Walsh
37 * Move creation of SecurityManager to main to prevent error on multiple
38 * calls to Launch
39 *
40 * Revision 0.3 - 9/05/96 6:38p Walsh
41 * Add a main method to handle command line agent launching
42 *
43 * Revision 0.2 - 9/04/96 12:35p Walsh
44 * Add some exception handling. Let ConduitServer handle details of
45 * launching agent. Let RemoteLoader build up the home codebase URL
46 *
47 * Revision 0.1 - 8/29/96 5:21p Walsh

```

- 5 -

```

Bootstrap.java_1      Tue Nov 12 07:44:47 1996      2

48   * initial version
49   *
50   */
51 package com.meitca.hsl.zonesagents.bootstrap;
52
53 import java.rmi.Naming;
54 import java.rmi.server.StubSecurityManager;
55 import java.rmi.server.MarshalOutputStream;
56 import java.net.InetAddress;
57 import java.util.*;
58 import java.io.*;
59
60 import com.meitca.hsl.zones.agents.shared.*;
61 import com.meitca.hsl.zones.agents.conduit.ConduitServer;
62 import com.meitca.hsl.zones.agents.conduit.AgentPackage;
63 import com.meitca.hsl.zones.agents.conduit.Itinerary;
64 import com.meitca.hsl.zones.agents.conduit.MobileCodebase;
65 import com.meitca.hsl.zones.agents.conduit.ConduitOutputStream;
66 import com.meitca.hsl.zones.agents.conduit.Destination;
67 import com.meitca.hsl.zones.agents.conduit.ConduitServerImpl;
68 import com.meitca.hsl.zones.agents.remoteloader.RemoteClassLoaderImpl;
69 import com.meitca.hsl.zones.agents.security.InsecureivityManager;
70
71 /**
72 */
73 /**
74  * Bootstrap provides a mechanism for boot strapping (launching) agents
75  * into the system. The calling program must provide the launch parameters
76  * in the form of the AgentLaunchInformation object. Bootstrap takes this
77  * information and provides the necessary steps to launch the indicated agents.
78  * This includes building creating the MobileCodebase, loading any needed
79  * classes (including the agent) into the MobileCodebase, constructing the
80  * AgentPackage, contacting the initial ConduitServer and sending the agent.
81  * @see AgentLaunchInformation
82  * @author thomas.walsh
83  */
84 public class Bootstrap {
85
86 /**
87  * Launches an agent.
88  * @param agent The agent to launch
89  * @param itinerary The agent's itinerary
90  * @param agentCodebaseURL An URL pointing to the codebase of the agent.
91  * This URL can take the following forms:
92  * <ul>
93  * <li>A file URL like <b>file:C:\agents</b> - which indicates that the
94  * agent's code is located on the local machine in the directory

```

- 51 -

```

95      *
96      * <b>C:\agent</b>
97      * like <b>http://hostname/agents</b> - which indicates a
98      * location on a web server from which the agents code can be
99      * retrieved. (This is very similar to the codebase modifier
100     * which can be added to an HTML <b>&tpplet</b> tag).
101     * <b>rmi://hostname/RemoteClassLoader</b> pointing
102     * to a RemoteClassLoader object which can retrieve the agents code.<br>
103     * This RMI URL could also look like
104     * <b>rmi://hostname/RemoteClassLoader/C:\agents</b> which indicates that
105     * the RemoteClassLoader on hostname should be used to retrieve
106     * the classes and that the agents code is located in the <b>C:\agents</b>
107     * directory.
108     * </ul>
109     * <caption>relatedClasses</caption> An array of fully qualified classnames of classes which this agent makes
110     * use of. These classes will be set up to travel with the agent.
111     * </p>
112     * <code>
113     * public static void launchAgent(Agent agent, Itinerary itinerary, String agentCodebaseURL,
114     * String[] relatedClasses) throws LaunchException {
115     *     try {
116     *         MobileCodebase agentCode = new MobileCodebase();
117     *         BootstrapClassLoader bootStrap = new BootstrapClassLoader(agentCodebaseURL,
118     *             agentCode);
119     *         bootStrap.loadAgentClass(agent.getClass().getName(), false);
120     *         if (relatedClasses != null) {
121     *             for (int i=0; i<relatedClasses.length; i++) {
122     *                 Class related = bootStrap.loadAgentClass(relatedClasses[i], false);
123             }
124         }
125         String absoluteCodebaseURL;
126         if (agentCodebaseURL == null) {
127             if (!agentCodebaseURL.startsWith("rmi:") && !agentCodebaseURL.startsWith("http:"))
128                 if (agentCodebaseURL.startsWith("file:")) {
129                     absoluteCodebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
130                         InetAddress.getLocalHost().getHostName(),
131                         String.format("%s", agentCodebaseURL));
132                 } else {
133                     absoluteCodebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
134                         InetAddress.getHostName(),
135                         agentCodebaseURL);
136             }
137         }
138     }
139 }

```

- 52 -

```

Bootstrap.java_1      Tue Nov 12 07:44:47 1996      4

137     } else {
138         absoluteCodebaseURL = agentCodebaseURL;
139     }
140 } else {
141     absoluteCodebaseURL = RemoteClassLoaderImpl.buildFmURL(
142         InetAddress.getLocalHost() .getLocalName());
143 }
144 AgentPackage agentPackage = new AgentPackage(agent,
145     Code,
146     rary,
147     uteCodebaseURL);
148 }
149 /**
150 * pass the agent along
151 */
152 CnduitServiceImpl.sendPackage(agentPackage);
153
154 } catch (Exception error) {
155     throw new LaunchException("LaunchException: " + error.getMessage());
156 }
157 }
158 /**
159 * Launches an agent based on the given launch parameters
160 * @param launchInfo The launch parameters
161 */
162 public static void launchAgent (AgentLaunchInformation launchInfo) throws LaunchException {
163
164     try {
165         MobileCodebase agentCode = new MobileCodebase();
166         BootStrapClassLoader bootStrap = new BootStrapClassLoader(launchInfo.itsAgentDirectory,
167             agentCode);
168
169         Class agentClass = bootStrap.loadAgentClassFromFile(launchInfo.itsAgentFile, true);
170
171         if (launchInfo.itsRelatedClasses != null ) {
172             for (int i=0; i<launchInfo.itsRelatedClasses.length; i++)
173                 Class related = bootStrap.loadAgentClassFromFile(launchInfo.itsRelatedClasses[i],
174                     true);
175
176         String codebaseURL = RemoteClassLoaderImpl.buildCodebaseURL(
177             InetAddress.getLocalHost() .getHostN

```

- 53 -

```

BootStrap.java_1      Tue Nov 12 07:44:47 1996      5

e(),
178
179
180
181     Agent agent;
182     if (launchInfo.itsAgent == null) {
183         agent = (Agent) agentClass.newInstance();
184     } else {
185         agent = launchInfo.itsAgent;
186     }
187     AgentPackage agentPackage = new AgentPackage(agent,
188
Code,
189     launchInfo.itsItinerary,
190
asURL());
191
192
193     // pass the agent along
194     ConduitServerImpl.sendPackage(agentPackage);
195
196     i catch (Exception error) {
197         throw new LaunchException("LaunchException: " + error.getMessage());
198     }
199
200
201 /**
202 * This method gets invoked when a user tries to launch an Agent from
203 * the command line. The command line should look something like the
204 * following:<br>
205 * <pre>
206 * Usage: bootstrap [-d hostname,method] [-f relatedfile.class] [-h] agentfile.class
207 *
208 * </pre><br>
209 * The user specifies the agent class file as the last parameter to
210 * the command. The destinations for the Agent are specified with the
211 * <b>-d</b> option. A destination is specified by the hostname follows by a
212 * comma (no space) followed by the name of the method to invoke.
213 * This could look something like the following: <b>-d host1,agentMethod1</b>.
214 * The user specifies related class files that should be sent with the
215 * agent using the <b>-f</b> option. Finally, the <b>-h</b> option brings up
216 * some command line help.
217 */
218
219 public static void main(String args[]) {
220
// Create and install the security manager

```

-54-

```

Tue Nov 12 07:44:47 1996      6

BootStrap.java_1

221
222     System.setSecurityManager(new InsecurityManager());
223
224     // We start by doing some rather brute force parsing of the
225     // command line. The command line args are used to build up
226     // an AgentLaunchInformation data structure.
227     AgentLaunchInformation launchInfo = new AgentLaunchInformation();
228     Vector destinationStrings = new Vector();
229     Vector relatedFiles = new Vector();
230     String agentFile = null;
231
232     for (int i=0; i<args.length; i++) {
233         if (args[i].equals("-d")) {
234             String destinationString = args[++i];
235             destinationStrings.addElement(destinationString);
236             continue;
237         } else if (args[i].equals("-f")) {
238             String fileName = args[++i];
239             relatedFiles.addElement(fileName);
240             continue;
241         } else if (args[i].equals("-h")) {
242             System.out.println(StringResources.USAGE);
243             return;
244         } else if (args[i].charAt(0) == '-') {
245             System.out.println(StringResources.ILLEGAL_);
246             System.out.println(StringResources.USAGE);
247             return;
248         } else {
249             agentFile = args[i];
250         }
251     }
252
253     // first lets verify that we have a valid agent file.
254     if (agentFile == null) {
255         System.out.println(StringResources.NO_AGENT);
256         System.out.println(StringResources.HELP);
257         return;
258     }
259
260
261     if (!agentFile.endsWith(".class")) {
262         System.out.println(StringResources.ILLEGAL_AGENT + " " +
263         System.out.println(StringResources.AGENT_HELP);
264         return;
265     }
266
267     launchInfo.itsAgentDirectory = new String(".");

```

BootStrap.java\_1

Tue Nov 12 07:44:47 1996 7

- 56 -

## BootStrap.java\_1

Tue Nov 12 07:44:47 1996 8

```

315     String fileName = (String)relatedFiles.elementAt(i);
316     if (fileName.endsWith(".class")) {
317         launchInfo.itsRelatedClasses[i] = fileName;
318     } else {
319         System.out.println(StringResources.ILLEGAL_RELFILE + fileName);
320         System.out.println(StringResources.RELFILE_HELP);
321         return;
322     }
323 }

324 // Finally, lets take a look at those destinations
325 if (destinationStrings.size() == 0) {
326     System.out.println(StringResources.NO_DEST);
327     return;
328 }
329

330 for (int i=0; i<destinationStrings.size(); i++) {
331     String destinationString = (String)destinationStrings.elementAt(i);
332     StringTokenizer tokenizer = new StringTokenizer(destinationString, " ,");
333     if (tokenizer.countTokens() != 2) {
334         System.out.println(StringResources.ILLEGAL_DEST + destinationString);
335         System.out.println(StringResources.USAGE);
336     }
337 }

338 String hostname = tokenizer.nextToken();
339 String methodName = tokenizer.nextToken();
340

341 // Find out if the methodname actually point to one of the methods
342 // in the Agent
343 int methodId = -1;
344 for(int j=0; j<launchInfo.itsMethods.length; j++) {
345     if (launchInfo.itsMethods[j].equals(methodName)) {
346         methodId = j;
347     }
348 }
349
350
351 if (methodId == -1) {
352     System.out.println(StringResources.ILLEGAL_METHOD + methodName);
353     System.out.println(StringResources.USAGE);
354
355     return;
356 }
357     launchInfo.itsItinerary.addDestination(new Destination(hostname, methodId));
358 }
359
360 // Now that we have parsed the command line and built up th

```

- 57 -

Bootstrap.java\_1

Tue Nov 12 07:44:47 1996 9

```
362 // launchInfo, we will actually attempt to launch the agent.
363 try {
364     launchAgent(launchInfo);
365 } catch (LaunchException e) {
366     System.out.println(StringResources.LAUNCH_ERROR + " " + e.getMessage());
367     e.printStackTrace();
368 }
369 }
370 }
```

- 58 -

```

BootstrapClassLoader.java_1      Tue Nov 12 07:44:47 1996      1
1  /* $Header: /com/meitca/hsl/zones/jagents/bootstrap/BootstrapClassLoader.java 3      11/11/96 5:13p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  /*
10 * DESCRIPTION
11 * ClassLoader used in agent bootstrap process.
12 * $Log: /com/meitca/hsl/zones/jagents/bootstrap/BootstrapClassLoader.java $
13 *
14 * 3  li/11/96 5:13p Billp
15 *     Made correction to company name.
16 *
17 * 2  9/28/96 6:56p Walsh
18 *     Use AgentCodebase object to retrieve bytecodes.
19 *
20 * 1  8/29/96 5:21p Walsh
21 *     initial version
22 *
23 */
24 package com.meitca.hsl.zones.jagents.bootstrap;
25
26 import java.io.*;
27 import java.net.*;
28 import java.rmi.Naming;
29
30 import com.meitca.hsl.zones.jagents.conduit.*;
31 import com.meitca.hsl.zones.jagents.shared.*;
32 import com.meitca.hsl.zones.jagents.remoteloader.*;
33
34 /**
35  * BootStrapClassLoader is used internally by the BootStrap
36  * class to load agent related classes
37  * @see Agent
38  * @see BootStrap
39  * @see MobileCodebase
40  * @author Thomas Walsh
41  */
42
43 public class BootstrapClassLoader extends ClassLoader {
44     /** An URL pointing to the codebase from which class files should be retrieved */
45     String itsHomeCodebaseURL;
46
47     /* An object representing the home codebase of the agent */

```

- 59 -

```

BootstrapClassLoader.java_1      Tue Nov 12 07:44:47 1996      2

48     AgentCodebase itsHomeCodebase;
49
50     /** The mobile code base into which to load classes */
51     MobileCodebase itsMobileCodebase;
52
53     /**
54      * object which actually handles the dirty work or retrieving .class
55      * files for a given class.
56     */
57     ClassFileLoader
58         itsClassFileLoader;
59
60     /**
61      * Constructs a BootStrapClassLoader.
62      * @param url The directory from which this ClassLoader should
63      * retrieve .class files
64     */
65     public BootStrapClassLoader(String url) throws MalformedURLException {
66         itsHomeCodebaseURL = url;
67         itsHomeCodebase = new AgentCodebase(url);
68         itsMobileCodebase = null;
69         itsClassFileLoader = new ClassFileLoader();
70     }
71
72     /**
73      * Constructs a BootStrapClassLoader. When this constructor is
74      * called, the ClassLoader realizes that their is an associated
75      * MobileCodebase object. As .class files are read from disk,
76      * they are also written into the MobileCodebase. Once they are
77      * loaded into the MobileCodebase, they are ready to travel.
78      * @param dir The directory from which this ClassLoader should
79      * retrieve .class files
80      * @param codename The associated MobileCodebase object
81      */
82     public BootStrapClassLoader(String url, MobileCodebase codebase) throws MalformedURLException {
83         this(url);
84         itsMobileCodebase = codebase;
85     }
86
87     /**
88      * Load a class. The classes .class file must be in the directory
89      * that was specified in the constructor. If a MobileCodebase
90      * was specified in the constructor, the class will also be loaded
91      * into that codebase.
92      * @param name The class name
93      * @param resolve True if resolveClass should be called
94     */

```

```

BootstrapClassLoader.java_1   Tue Nov 12 07:44:47 1996   3
95  public Class loadAgentClass(String name, boolean resolve) throws ClassNotFoundException {
96
97      byte[] bytecodes = itsHomeCodebase.retrieveCode(name);
98
99      Class c = defineClass(bytecodes, 0, bytecodes.length);
100     if (resolve)
101         resolveClass(c);
102
103     if (itsMobileCodebase != null)
104         itsMobileCodebase.storeCode(c.getName(), bytecodes);
105
106     return c;
107 }
108
109 /**
110  * Load a class from the specified file. The file must be in the
111  * directory that was specified in the constructor. If a
112  * MobileCodebase was specified in the constructor, the class will
113  * also be loaded into that codebase.
114  * @param name The class filename
115  * @param resolve True if resolveClass should be called
116  */
117 //STUBABILITY: Integrate this method with loadAgentClass
118 public Class loadAgentClassFromFile(String fileName, boolean resolve)
119     throws FileNotFoundException, IOException, ClassNotFoundException , ClassFormatError;
120
121     File file = new File(itsHomeCodebaseURL, fileName);
122     long length = file.length();
123     FileInputStream stream = new FileInputStream(file);
124     byte data[] = new byte((int)length);
125
126     stream.read(data);
127     Class c = defineClass(data, 0, (int)length);
128     if (resolve) {
129         resolveClass(c);
130     }
131
132     if (itsMobileCodebase != null)
133         itsMobileCodebase.storeCode(c.getName(), data);
134
135     return c;
136
137 /**
138  * The java.lang.ClassLoader version of loadClass.
139  * First locks for a standard system class, and then looks for an
140  * agent class in the directory that was specified in the constructor.
141  * If the class is an Agent related class, then its bytecodes are

```

BootStrapClassLoader.java\_1      Tue Nov 12 07:44:47 1996      4

```
142        * loaded into the NobileCodebase
143        * @param      name The class name
144        * @param      resolve true if resolveClass should be called
145        */
146        protected Class loadClass(String name, boolean resolve) throws ClassNotFoundException {
147            Class c = findSystemClass(name);
148            if (resolve) {
149               resovleClass(c);
150            }
151            return c;
152        }
153        } catch (ClassNotFoundException ei) {
154        try {
155            return loadAgentClass(name, resolve);
156        } catch (Exception e2) {
157            throw new ClassNotFoundException("Could not load " + name);
158        }
159        }
160        }
161 }
```

- 62 -

```

FindFilePanel.java_1      Tue Nov 12 07:44:47 1996      1
1  /*
2   * $Header: /com/meitca/hsl/zonesagents/bootstrap/FindFilePanel.java 5      11/11/96 5:13p Billp $
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   * DESCRIPTION
10  * The FindFilePanel is the AgentLaunchWizard
11  *
12  * $Log: /com/meitca/hsl/zonesagents/bootstrap/FindFilePanel.java 5      11/11/96 5:13p Billp
13  *
14  * 5      11/11/96 5:13p Billp
15  * Made correction to company name.
16  *
17  * 4      10/07/96 4:44p Walsh
18  * Save name & location of last agent launched. Restart Wizard printing
19  * at same agent.
20  *
21  * 3      9/10/96 11:46a Walsh
22  * remove bogus imports that javac considers illegal
23  *
24  * 2      9/09/96 3:45p Walsh
25  * replace gjt.BrowserDialog w/ com.meitca.hsl.util.InfoDialog
26  *
27  * 1      8/29/96 5:21p Walsh
28  * initial version
29  *
30  */
31 package com.meitca.hsl.zonesagents.bootstrap;
32
33 import java.awt.*;
34 import java.awt.*;
35 import gjt.awt.*;
36
37 import com.meitca.hsl.util.Wizard;
38 import com.meitca.hsl.util.WizardPanel;
39 import com.meitca.hsl.util.InfoDialog;
40
41 /**
42  * the first panel of the AgentLaunchWizard. This panel allows the user
43  * to specify the file-name of the .class file containing the agent.
44  * @see AgentLaunchWizardPanel
45  * @see AgentLaunchWizardPanel
46  * @see AgentLaunchWizardPanel
47  * @author Thomas Walsh

```

FindfilePanel.java\_1 Tue Nov 12 07:44:47 1996 2

```

48  */
49  public class FindfilePanel extends AgentLaunchwizardPanel {
50      /** The edit field */
51      JTextField itsFileBrowseField;
52
53      /** The "Browse..." button */
54      JButton itsFileBrowseButton;
55
56
57      /**
58       * Construct the Findfilepanel and all of its components.
59       * @param wizard The wizard to which this panel belongs (needed by
60       * @param super)
61       * @param wiinfoWizard The AgentLaunchInformation describing this
62       * agent launch (needed by super)
63
64     public FindfilePanel(WizardPanel wizard, AgentLaunchInformation info) {
65         super(wizard, info);
66         buildPanel();
67     }
68
69     /**
70      * Verify that the information entered into the panel is valid. Called
71      * after the "Next >" button is pressed.
72
73     /**
74      * The method validates the following things...
75      *
76      * 1) A class file name is specified
77      * 2) The file name ends with ".class"
78      * 3) The file exists
79      * 4) The file contains valid Java bytecodes
80      * 5) The file contains a class derived from Agent
81      * 6) An invocation skeleton is available
82      *    since if the panel contains valid info, else if not
83
84     public boolean validatePanel() {
85         /*
86          * we want to break the full filename up into the directory portion
87          * and the filename portion. Find the last occurrence of the
88          * directory separator character
89          */
90         String fullfilename = itsFileBrowseField.getText();
91
92         /*
93          * Verify that someone typed something into the edit control and that
94          * what they typed in ends with ".class"
95          */
96         if (fullName.length() == 0) {
97             InfoDialog info = new InfoDialog(itsWizard,
98                     StringResources.LAUNCH_ERROR_TITLE,
99                     StringResources.StringResources.lightGray);
100            info.setBackground(Color.lightGray);
101            info.show();
102        }

```

- 6 -

```

FindfilePanel.java_1      Tue Nov 12 07:44:47 1996      3

95
96      return false;
97
98      if (!fullName.endsWith(".class")) {
99          InfoDialog info = new InfoDialog(itsWizard,
100          StringResources.LAUNCH_ERROR_TITLE,
101          info.setBackground(Color.lightGray);
102          info.show());
103          return false;
104      }
105
106      itsAgentInfo.itsAgentDirectory = new String("");
107      itsAgentInfo.itsAgentDirectory = fullName;
108
109      for (int i=fullName.length()-1; i>=0; i--) {
110          if (fullName.charAt(i) == File.separatorChar) {
111              itsAgentInfo.itsAgentDirectory = fullName.substring(0, i+1);
112              itsAgentInfo.itsAgentFile = fullName.substring(i+1, fullName.length());
113              break;
114          }
115      }
116
117      // Now let the AgentClassInfoGatherer verify the existence of the file,
118      // that the file contains a valid Java Agent and that the skelton
119      // exists.
120      try {
121          AgentClassInfoGatherer agentInfoGatherer = new AgentClassInfoGatherer(
122              itsAgentInfo.itsAgentDirectory, itsAgentInfo.itsAgentFile);
123
124          itsAgentInfo.itsMethods = agentInfoGatherer.getAgentMethods();
125
126          return true;
127      } catch (FileNotFoundException e1) {
128          InfoDialog info = new InfoDialog(itsWizard,
129          StringResources.LAUNCH_ERROR_TITLE,
130          info.setBackground(Color.lightGray));
131          info.show();
132          return false;
133      } catch (IOException e1) {
134          InfoDialog info = new InfoDialog(itsWizard,
135          StringResources.LAUNCH_ERROR_TITLE,
136          StringResources.IOERROR_TITLE,
137          info.setBackground(Color.lightGray));
138          info.show();
139          return false;
140      } catch (ClassNotFoundException e1) {
141

```

- 65 -

**FindfilePanel.java\_1****Tue Nov 12 07:44:47 1996 4**

```

142
143   InfoDialog info = new InfoDialog(itsWizard,
144                                     StringResources.LAUNCH_ERROR_TITLE,
145                                     StringResources.BAD_CLASS_FORMAT);
146   info.setBackground(Color.lightGray);
147   info.show();
148   return false;
149 } catch (NoAgentException e1) {
150   InfoDialog info = new InfoDialog(itsWizard,
151                                     StringResources.LAUNCH_ERROR_TITLE,
152                                     StringResources.NOT_AGENT);
153   info.setBackground(Color.lightGray);
154   info.show();
155 } catch (ClassNotFoundedException e1) {
156   InfoDialog info = new InfoDialog(itsWizard,
157                                     StringResources.LAUNCH_ERROR_TITLE,
158                                     StringResources.BAD_CLASS_FORMAT);
159   info.setBackground(Color.lightGray);
160   info.show();
161   return false;
162 } catch (NoAgentSkeletonException e1) {
163   InfoDialog info = new InfoDialog(itsWizard,
164                                     StringResources.LAUNCH_ERROR_TITLE,
165                                     StringResources.NO_SKELTON);
166   info.setBackground(Color.lightGray);
167   info.show();
168   return false;
169 }
170 }
171 /**
172 * Builds the GUI Components for this panel
173 */
174 private void buildPanel() {
175   setLayout(new GridBagLayout());
176   constraint(this, label1,
177             Label label1 = new Label(StringResources.FF_TEXT1, label.LEFT);
178   constraint(this, label1,
179             constraint(this, label1,
180             Label label2 = new Label(StringResources.FF_TEXT2, label.LEFT);
181   constraint(this, label2,
182             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0
183   constraint(this, label2,
184             0, 1, 1,
185             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0
186   constraint(this, label2,
187             0, 1, 1,
188             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0
189

```

5

```

189 Label label3 = new Label(StringResources.FF_TEXT3, Label.LEFT);
190 constrain(this, label3, 0, 2, 1, 1,
191 GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
192 0.0, 0.0);
193
194 Label label4 = new Label(StringResources.FF_TEXT4, Label.LEFT);
195 constrain(this, label4, 0, 3, 1, 1,
196 GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
197 0.0, 0.0);
198
199 Panel browsePanel = new Panel();
200 constrain(this, browsePanel,
201 0, 4, 1, 1,
202 GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 1.0, 1.0,
203 0.0, 0.0);
204
205 browsePanel.setLayout(new GridLayout());
206 Label findFileLabel = new Label(StringResources.FF_FIELD_LABEL, Label.LEFT);
207 constrain(browsePanel, findFileLabel,
208 1, 0, 1, 1,
209 GridBagConstraints.NONE, GridBagConstraints.NORTHWEST, 0.0, 0.0,
210 0.0, 0.0);
211
212 itsFileBrowseField = new TextField(25);
213 constrain(browsePanel, itsFileBrowseField,
214 2, 0, 1, 1,
215 GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 1.0, 1.0,
216 0.0, 0.0);
217
218 itsFileBrowseButton = new Button(StringResources.FF_BROWSE_BUTTON);
219 constrain(browsePanel, itsFileBrowseButton,
220 3, 0, 1, 1,
221 GridBagConstraints.NONE, GridBagConstraints.NORTHWEST, 0.0, 0.0,
222 0.0, 0.0);
223
224 /**
225 * Derived from Component.action */
226 public boolean action(Event event, Object what) {
227 if (event.target == itsFileBrowseButton) {
228   doBrowse();
229   return true;
230 }
231 return false;
232 }
```

```

FindfilePanel.java_1      Tue Nov 12 07:44:47 1996      6

236
237
238     /**
239     * Called in response to selecting the Panel's "Browse..." button. Brings up
240     * a Open File Dialog.
241     */
242     private void doBrowse() {
243         FileDialog browse = new FileDialog(Util.getFrame(this), StringResources.BROWSE_DLG_TITLE);
244         browse.setFile("*.class");
245
246         if (itsAgentInfo.itsAgentDirectory != null) {
247             if (itsAgentInfo.itsAgentDirectory.length() != 0) {
248                 browse.setDirectory(itsAgentInfo.itsAgentDirectory);
249             }
250         }
251
252         browse.show();
253
254         String fullfilename = new String("");
255
256         if (browse.getDirectory() != null) {
257             if (browse.getDirectory().length() != 0) {
258                 fullfilename = browse.getDirectory();
259             }
260         }
261
262         if (browse.getFile() != null) {
263             if (browse.getFile().length() != 0) {
264                 fullfilename = fullfilename + browse.getFile();
265             }
266         }
267
268         if (!fullfilename.length() != 0)
269             itsFileBrowseField.setText(fullfilename);
270
271
272     /**
273     * initializes the panel for display
274     */
275     public void displayPanel() {
276         String fullfilename = "";
277
278         if (itsAgentInfo.itsAgentDirectory != null) {
279             if (itsAgentInfo.itsAgentDirectory.length() != 0) {
280                 fullfilename = itsAgentInfo.itsAgentDirectory;
281
282                 if (!fullfilename.endsWith(File.separator)) {

```

```
FindfilePanel.java_1      Tue Nov 12 07:44:47 1996      7

283                               fullFileName += File.separator;
284                               )
285                               )
286                           )
287
288                           if (itsAgentInfo.itsAgentFile != null) {
289                               if (itsAgentInfo.itsAgentFile.length() != 0) {
290                                   fullFileName += itsAgentInfo.itsAgentFile;
291                               }
292                           }
293
294                           if (fullFileName.length() != 0)
295                               itsFileBrowseField.setText (fullFileName);
296
297
298
299
300 }
```

```

ItineraryGridDataModel.java_1      Tue Nov 12 07:44:47 1996   1
1  /*
2   * $Header: /com/meitca/hsl/zonesagents/bootstrap/ItineraryGridDataModel.java 3 11/11/96 5:13p
3   *
4   * Copyright 1996 Horizon Systems Laboratory,
5   * Mitsubishi Electric Information Technology Center, America.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC INC.
8   */
9   *
10  * DESCRIPTION:
11  * A Data Model class for use in conjunction with the Rogue Wave
12  * Grid class.
13  *
14  * $Log: /com/meitca/hsl/zonesagents/bootstrap/ItineraryGridDataModel.java $
15  * 3 11/11/96 5:13p Billp
16  * Made correction to company name.
17  *
18  * 2 9/04/96 12:26p Walsh
19  * Propagate changes to Destination terminology
20  *
21  * 1 8/29/96 5:21p Walsh
22  * initial version
23  *
24  */
25 package com.meitca.hsl.zonesagents.bootstrap;
26
27 import com.roguewave.widgets.*;
28 import com.roguewave.widgets.grid.*;
29 import java.util.Vector;
30 import java.utilEnumeration;
31
32 import com.meitca.hsl.zonesagents.conduit.Destination;
33 import com.meitca.hsl.zonesagents.conduit.Itinerary;
34
35 /**
36  * Implements the necessary com.roguewave.widgets.* interfaces to
37  * support the display of a Grid object. This class is tied to the
38  * Grid control on the ItineraryPanel of the AgentLaunchWizard. The
39  * grid is used to specify the Itinerary of the Agent to launch
40  * @see
41  * @see
42  * @see
43  * @author
44  */
45 public class ItineraryGridDataModel extends SelectionGridModel {
46  /** The rows displayed in the grid */
47  Vector itsRows;

```

-70-

```

ItinerarygridDataModel.java_1 Tue Nov 12 07:44:47 1996 2

48 	/** The ItineraryPanel this object is tied to. */
49 	ItineraryPanel itsPanel;
50
51 /**
52  * Construct a ItineraryGridDataModel
53  * @param panel The ItineraryPanel this object is tied to.
54  */
55 public ItineraryGridDataModel(ItineraryPanel panel) {
56 	itsRows = new Vector();
57 	itsPanel = panel;
58 }
59

60 /**
61  * derived from SelectionGridModel.fetchString */
62 	public String fetchString(int row, int column) {
63 	try {
64
65 // vectors are zero based...Grids are one based, so we need to
66 // subtract one
67 	Destination aRow = (Destination) itsRows.elementAt(row-1);
68 	switch (column) {
69 	case 1:
70 // return aRow.getDestinationHost();
71 	default:
72 // return "";
73 	}
74 } catch (ArrayIndexOutOfBoundsException e){
75 	return "";
76 }
77 /**
78  * derived from SelectionGridModel.fetchInteger */
79 	public int fetchInteger (int row, int column) {
80 	try {
81
82 // vectors are zero based...Grids are one based, so we need to
83 // subtract one
84 	Destination aRow = (Destination) itsRows.elementAt(row-1);
85 	switch (column) {
86 	case 2:
87 // return aRow.getMethodID() + 1;
88 	default:
89 // return 0;
90 	}
91 } catch (ArrayIndexOutOfBoundsException e){
92 	return 0;
93 }
94

```

```

itineraryGridDataModel.java_1    Tue Nov 12 07:44:47 1996      3

95     /** derived from SelectionGridModel.update */
96     public boolean update (int row, int column, String value) {
97         aRow;
98
99         try {
100             // vectors are zero based... Grids are one based, so we need to
101             // subtract one
102             aRow = (Destination) itsRows.elementAt (row-1);
103
104             } catch (ArrayIndexOutOfBoundsException e) {
105                 aRow = new Destination();
106
107                 itsRows.insertElementAt (aRow, i + x-1);
108
109                 // new destination has been added to the Itinerary. Lets add
110                 // a new blank row to the Grid so that the user has space to type any
111                 // more destinations
112                 itsPanel.addRow();
113
114             switch (column) {
115                 case 1:
116                     aRow.setDestinations' (value);
117                     break;
118
119                     default:
120                         break;
121
122             }
123
124             // derived from SelectionGridModel.update
125             public boolean update (int row, int column, int value) {
126                 aRow;
127
128                 try {
129                     // vectors are zero based... Grids are one based, so we need to
130                     // subtract one
131                     aRow = (Destination) itsRows.elementAt (row-1);
132                     aRow = new Destination();
133
134                     itsRows.insertElementAt (aRow, row-1);
135
136                     // new destination has been added to the Itinerary. Lets add
137                     // a new blank row to the Grid so that the user has space to type any
138                     // more destinations
139                     itsPanel.addRow();
140
141             switch (column) {

```

```
ItineraryGridDataModel.java_1      Tue Nov 12 07:44:47 1996    4
142
143           case 2:
144           aRow.setMethodID(value - 1);
145           break;
146       default:
147           break;
148       }
149   }
150
151   /**
152    * Retrieves the Itinerary which has been specified by the user.
153   */
154   public Itinerary getItinerary() {
155       Itinerary itinerary = new Itinerary();
156       Enumeration enum = itsRows.elements();
157
158       while(enum.hasMoreElements()) {
159           Destination destination = (Destination)enum.nextElement();
160           itinerary.addDestination(destination);
161
162       }
163   }
164 }
```

- 73 -

```

ItineraryPanel.java_1      Tue Nov 12 07:44:47 1996      1
1  /*
2   * $Header: /com/meitca/hsl/zonesagents/bootstrap/ItineraryPanel.java 3      11/11/96 5:13p Billp $
3   *
4   * Copyright 1996 Horizon Systems Laboratory,
5   * Mitsubishi Electric Information Technology Center America.
6   * All rights reserved.
7   *
8   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
9   *
10  * DESCRIPTION
11  * The ItineraryPanel of the AgentLaunchWizard
12  *
13  * $Log: /com/meitca/hsl/zonesagents/bootstrap/ItineraryPanel.java $
14  * 3 11/11/96 5:13p Billp
15  *     Made correction to company name.
16  *
17  * 2 9/10/96 11:46a Walsh
18  *     remove bogus imports that javac considers illegal
19  *
20  * 1 8/29/96 5:21p Walsh
21  *     initial version
22  *
23  */
24 package com.meitca.hsl.zonesagents.bootstrap;
25 import java.awt.*;
26 import java.io.*;
27 import com.mrcwave.widgets.grid.*;
28
29 import com.meitca.hsl.util.Wizard;
30 import com.meitca.hsl.util.WizardPanel;
31
32 /**
33 * The third panel of the AgentLaunchWizard. This panel allows the user
34 * to specify the itinerary of the agent.
35 * @see Itinerary
36 * @see AgentLaunchWizardPanel
37 * @see AgentLaunchWizard
38 * @author Thomas Walsh
39 */
40
41 public class ItineraryPanel extends AgentLaunchWizardPanel {
42     /** The grid control on the panel */
43     Grid itsGrid;
44
45     /** The date model object used in conjunction with itsGrid */
46     ItineraryGridDataModel itsDataModel;
47

```

```

ItineraryPanel.java_1      Tue Nov 12 07:44:47 1996      2

48  /**
49   * Construct the ItineraryPanel and all of its components.
50   * @param wizard The wizard to which this panel belongs (needed by
51   * sure)
52   * @param infoinforntation describing this
53   * agent launch (needed by Agent)
54   */
55  public ItineraryPanel(Wizard wizard, AgentLaunchInformation info) {
56      super(wizard, info);
57      buildPanel();
58  }
59
60  /**
61   * The wizard calls this method right before it is going to display
62   * the panel. In this case, the panel needs to recreate the Grid
63   * so that the ChoiceColumn of the grid contains the names of the
64   * available methods of the currently selected Agent.
65   */
66  public void displayPanel() {
67      remove(itsGrid);
68
69      // Create a new grid model
70      itsDataModel = new ItineraryGridDataModel(this);
71
72      // Create a new Grid in spreadsheet mode
73      itsGrid = new Grid(Grid.MODE_TABLE_MODE, itsDataModel, itsDataModel);
74      itsGrid.getCellPanel().setCellColor(Color.gray);
75      itsGrid.getCellPanel().setHorzLineColor(Color.gray);
76      itsGrid.setEditable(true);
77
78      // Set the grid column heading panel's attributes //
79      itsGrid.setColumnHeadingPanel(new ColumnHeadingPanel());
80      itsGrid.getColumnHeadingPanel().setBackground(Color.lightGray);
81      itsGrid.getColumnHeadingPanel().setForeground(Color.black);
82      itsGrid.getColumnHeadingPanel().setTreeLineStyle(Color.gray);
83      itsGrid.getColumnHeadingPanel().setHorzLines(true);
84      itsGrid.getColumnHeadingPanel().setTextLines(1);
85
86      // Add a text field column to the grid //
87      TextFieldColumnAttribute coll;
88      coll.setTitle("StringResources.DEST_COL_TITLE");
89      coll.setWidth(150);
90      ((TableColumnCollection) itsGrid.getGridColumnCollection()).add(coll);
91
92      ChoiceColumnAttribute choiceColumn = new ChoiceColumnAttribute();
93      choiceColumn.setTitle(StringResources.ACTION_COL_TITLE);
94      choiceColumn.setWidth(150);

```

- 75 -

```

ItineraryPanel.java_1      Tue Nov 12 07:44:47 1996      3

95   ((TableColumnCollection) itsGrid.getColumnCollection()).add(choiceColumn);
96
97   // Set the grid to have 1 row.  //
98   (itsGrid.getRowCollection()).setRows(1);
99
100  constraint(this, itsGrid,
101      0, 6, 1, 1,
102      GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
103      0.0, 0.0);
104
105
106  choiceColumn.addItem("");
107  for(int i=0; i<itsAgentInfo.itsMethods.length; i++) {
108      choiceColumn.addItem(itsAgentInfo.itsMethods[i]);
109  }
110
111
112  invalidate();
113  validate();
114
115
116  /**
117   * Verify that the information entered into the panel is valid.  Called
118   * after the "Next >" button is pressed.
119   * There is no actual verification necessary.  This method simply
120   * retrieves the itinerary from the data model and stuffs it into
121   * the AgentLaunchInfo.
122   * & returns true if the panel contains valid info, false if not
123   */
124  public boolean validatePanel() {
125      itsAgentInfo.setItinerary = itsDataModel.getItinerary();
126      return true;
127  }
128
129
130  /**
131   * Builds the GUI components for this panel
132  */
133  private void buildPanel() {
134      setLayout(new GridBagLayout());
135
136      Label label1 = new Label(StringResources.ITIN_TEXT1, Label.LEFT);
137      constraint(this, label1,
138          0, 0, 1, 1,
139          GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
140
141      label2 = new Label(StringResources.ITIN_TEXT2, Label.LEFT);

```

- 76 -

```

ItineraryPanel.java_1      Tue Nov 12 07:44:47 1996   4

142     constrain(this, label2, 0, 1, 1, 1,
143           GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
144           0.0, 0.0);
145
146
147
148     Label label3 = new Label(StringResources.ITIN_TEXT3, Label.LEFT);
149     constrain(this, label3, 0, 2, 1, 1,
150           GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
151           0.0, 0.0);
152
153
154     Label label4 = new Label(StringResources.ITIN_TEXT4, Label.LEFT);
155     constrain(this, label4, 0, 3, 1, 1,
156           GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
157           0.0, 0.0);
158
159
160     Label label5 = new Label(StringResources.ITIN_TEXT5, Label.LEFT);
161     constrain(this, label5, 0, 4, 1, 1,
162           GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
163           0.0, 0.0);
164
165
166     Label label6 = new Label(StringResources.ITIN_TEXT6, Label.LEFT);
167     constrain(this, label6, 0, 5, 1, 1,
168           GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0.0, 0.0,
169           0.0, 0.0);
170
171
172
173     /* Create a new grid model
174     itsDataModel= new ItineraryGridDataModel(this);
175     /* Create a new Grid in spreadsheet mode.
176     itsGrid = new Grid(Grid.TABLE_MODE,itsDataModel,itsDataModel,itsDataModel);
177     itsGrid.getCellPanel().setVertLineColor(Color.gray);
178     itsGrid.getCellPanel().setHorzLineColor(Color.gray);
179     itsGrid.setEditable = true;
180
181     /* Set the grid column heading panel's attributes //
182     itsGrid.setColumnHeadingPanel(new ColumnHeadingPanel());
183     itsGrid.getColumnHeadingPanel().setBackground(Color.lightGray);
184     itsGrid.getColumnHeadingPanel().setForeground(Color.black);
185     itsGrid.getColumnHeadingPanel().setThreeDStyle(Grid.THREE_D_RAISED);
186     itsGrid.getColumnHeadingPanel().setHorzLines(true);
187     itsGrid.getColumnHeadingPanel().setTextLines(1);
188

```

```

ItineraryPanel.java_1      Tue Nov 12 07:44:47 1996      5
189   // Add a text field column to the grid //
190   textFieldOldColumnAttribute coll;
191   coll = new JTextFieldAttribute();
192   coll.setCellColumnAttribute(DEF_COL_TITLE);
193   coll.setWidth(150);
194   ((TableColumnCollection) itsGrid).addColumnDefinition().add(coll);
195   choiceColumn.setAttribute(choiceColumn.getAttribute().add(zoi));
196   choiceColumn.setTitle(new ChoiceColumnAttribute());
197   choiceColumn.setTitle(StringResources.ACTION_COL_TITLE);
198   choiceColumn.setWidth(150);
199   ((TableColumnCollection) itsGrid.getGridColumnCollection()).add(choiceColumn);
200
201   // Set the grid to have 1 row.  /*
202   (itsGrid.getRowCollection()).setRows(1);
203
204   constraint(this, itsGrid,
205
206   207   6, 6, 1, 1,
207   GridBagConstraints.NORTH, GridBagConstraints.NORTHWEST, 1, 0, 1, 0,
208   209   0, 0, 0, 0);
210
211   */
212
213   /**
214   * called by the itineraryGrid:Model to signify that a new row should be
215   * added to the grid
216   */
217   protected void addRow() {
218     itsGrid.getRowCollection().setRows(itsGrid.getRowCollection().count() + 1);
219   }
220

```

- 78 -

ItineraryPanel.java\_1        Tue Nov 12 07:44:47 1996        5

```
199     // Add a text field column to the grid //
199     TextFieldColumnAttribute coll;
199     coll = new TextFieldColumnAttribute();
199     coll.setTitle(StringResources.DEST_COL_TITLE);
199     coll.setWidth(150);
199     ((TableColumnCollection) itsGrid.getColumnCollection()).add(coll);
199
199     ChoiceColumnAttribute choiceColumn = new ChoiceColumnAttribute();
199     choiceColumn.setTitle(StringResources.ACTION_COL_TITLE);
199     choiceColumn.setWidth(150);
199     choiceColumn.getCollection().add(choiceColumn);
199     ((TableColumnCollection) itsGrid.getColumnCollection()).add(choiceColumn);
199
200
201     // Set the grid to have 1 row. //
202     (itisGrid.getRowCollection()).setRow(1);
203
204     constraintThis. itsGrid. 0, 6, 1, 1,
205                      GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1.0, 1.0,
206                      0.0, 0.0;
207                      0.0, 0.0;
208
209
210
211
212
213     /*
213     * called by the ItineraryGridDataModel to signify that a new row should be
213     * added to the Grid
213     */
214
215     protected void addRow() {
216         itsGrid.getRowCollection().setRows(itsGrid.getRowCollection().count() + 1);
217
218         }
219
220 }
```

- 75 -

## NoAgentSkeletonException.java\_1      Tue Nov 12 07:44:48 1996

```

1  /*
2   * $Header: /ccm/meitca/hs1/zonesagents/bootstrap/NoAgentSkeletonException.java 2 11/11/96 5:13p Billp $
3   *
4   * Copyright 1996 Horizon Systems Laboratory,
5   * Mitsubishi Electric Information Technology Center America.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   * DESCRIPTION
10  * Exception indicating that Agent invocation Skeleton is missing.
11  *
12  * $Log: /cvm/meitca/hs1/zonesagents/bootstrap/NoAgentSkeletonException.java $
13  *
14  * 2 11/11/96 5:13p Billp
15  * Made correction to company name.
16  *
17  * 1 8/29/96 5:21p Walsh
18  * initial version
19  *
20  */
21 package com.meitca.hs1.zonesagents.bootstrap;
22
23 /**
24  * Exception indicates that the system could not locate the Agent
25  * method invocation skeleton. This skeleton is used by the conduit
26  * server to invoke the proper method on the Agent when it arrives
27  * at a destination. The skeleton is generated by the Agent Skeleton
28  * Generator.
29  * @see Agent
30  * @see Agentskeleton
31  * @author Thomas Walsh
32  */
33 public class NoAgentSkeletonException extends Exception {
34     public NoAgentSkeletonException() {
35         super();
36     }
37
38     public NoAgentSkeletonException(String s) {
39         super(s);
40     }
41 }

```

- 80 -

```

NotAgentException.java_1      Tue Nov 12 07:44:48 1996      1
1  /*
2   *  SHeader: /com/meitca/hsl/zonesagents/bootstrap/NotAgentException.java 3      11/11/96 5:13p Billp $
3   *  Copyright 1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *  All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   */
9   * DESCRIPTION
10  * Exception indicating that a .class file does not contain an Agent.
11  */
12  * $Log: /com/meitca/hsl/zonesagents/bootstrap/NotAgentException.java $
13  *
14  * 3 11/11/96 5:13p Billp
15  * Made correction to company name.
16  */
17  * 2 9/30/96 6:35p Welsh
18  * fix up javadoc comments
19  */
20  * 1 8/29/96 5:21p Walsh
21  * initial version
22  */
23  */
24 package com.meitca.hsl.zonesagents.bootstrap;
25 /**
26 */
27 * Exception indicates the class file specified did not contain
28 * a java class derived from Agent.
29 * @see Agent
30 * @author Thomas Welsh
31 */
32 public class NotAgentException extends Exception {
33     public NotAgentException() {
34         super();
35     }
36
37     public NotAgentException(String s) {
38         super(s);
39     }
40 }
```

```

relatedFilesPanel.java_1 Tue Nov 12 07:44:48 1996
1  /*
2   * Header: /com/meitca/hsl/zonesagents/bootstrap/RelatedFilesPanel.java 3
3   * Copyright 1996 Horizon Systems Laboratory.
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   */
9   * DESCRIPTION:
10  * The RelatedFilesPanel of the AgentLaunchWizard
11  */
12  */
13  */
14  * 3 11/11/96 5:13p BillP
15  * Made correction to company name.
16  */
17  * 2 9/10/95 11:46a Walsh
18  * remove bogus imports that javac considers illegal
19  */
20  * 1 8/29/95 5:21p Walsh
21  * initial version
22  */
23  */
24 package com.meitca.hsl.zonesagents.bootstrap;
25
26 import java.awt.*;
27 import javax.swing.*;
28
29 import com.meitca.hsl.util.Wizard;
30 import com.meitca.hsl.util.WizardPanel;
31
32 /**
33 */
34 /**
35 * The second panel of the AgentLaunchWizard. This panel allows the user
36 * to specify the filenames of classes that should be sent with the agent.
37 * @see Agent
38 * @see AgentLaunchWizardPanel
39 * @see AgentLaunchWizard
40 * @author Thomas Walsh
41 public class RelatedFilesPanel extends AgentLaunchWizardPanel {
42 /**
43 * The list control form which the user selects files
44 * List itsRelatedClassesList;
45
46 /**
47 * Construct the FindFilePanel and all of its components.

```

- 82 -

```

RelatedFilesPanel.java_1      Tue Nov 12 07:44:48 1996      2

48     * @param      wizard The wizard to which this panel belongs (needed by
49     *           super)
50     * @param      winfozard The AgentLaunchInformation describing this
51     *           agent launch (needed by super)
52   */
53   public RelatedFilesPanel(Wizard wizard, AgentLaunchInformation info) {
54     super(wizard, info);
55     buildPanel();
56   }
57
58 /**
59  * The Wizard calls this method right before it is going to display
60  * the panel. In this case, the Panel needs to fill the List
61  * with the names of all the .class files in the same directory as
62  * the Agent.
63 */
64 public void displayPanel() {
65   File agentDir = new File(itsAgentInfo.itsAgentDirectory);
66   String[] files = agentDir.list(new ClassFileFilter());
67
68   itsRelatedClassesList.clear();
69   for (int i=0; i<files.length; i++) {
70     if (files[i].equals(itsAgentInfo.itsAgentFile))
71       itsRelatedClassesList.addItem(files[i]);
72   }
73 }
74
75 /**
76  * Verify that the information entered into the panel is valid. Called
77  * after the "Next >" button is pressed.
78  * There is no actual verification necessary. This method simply
79  * retrieves the list of classes and stuffs them into the
80  * AgentLaunchInfo
81  * @return      true if the panel contains valid info, false if not
82 */
83 public boolean validatePanel() {
84   itsAgentInfo.itsRelatedClasses = itsRelatedClassesList.getSelectedItems();
85   itsAgentInfo.itsRelatedClasses = itsRelatedClassesList.getSelectedItems();
86   return true;
87 }
88
89 /**
90  * Builds the GUI Components for this panel
91  */
92 private void buildPanel() {
93   setLayout(new GridLayout());
94

```

```

RelatedFilesPanel.java_1      Tue Nov 12 07:44:48 1996      3
                                : 83 :
95     Label label11 = new Label(StringResources.RF_TEXT1, Label.LEFT);
96         constrain(this, label11, 0, 0, 1, 1,
97                     0, 0, 0, 0);
98             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
99
100    Label label12 = new Label(StringResources.RF_TEXT2, Label.LEFT);
101        constrain(this, label12, 0, 1, 1,
102                     0, 0, 0, 0);
103             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
104
105    Label label13 = new Label(StringResources.RF_TEXT3, Label.LEFT);
106        constrain(this, label13, 0, 2, 1, 1,
107                     0, 0, 0, 0);
108             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
109
110    Label label14 = new Label(StringResources.RF_TEXT4, Label.LEFT);
111        constrain(this, label14, 0, 3, 1, 1,
112                     0, 0, 0, 0);
113             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
114
115    Label label15 = new Label(StringResources.RF_TEXT5, Label.LEFT);
116        constrain(this, label15, 0, 4, 1, 1,
117                     0, 0, 0, 0);
118             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
119
120    Label label16 = new Label(StringResources.RF_TEXT6, Label.LEFT);
121        constrain(this, label16, 0, 5, 1, 1,
122                     0, 0, 0, 0);
123             GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTHWEST, 0, 0, 0, 0,
124
125    itsRelatedClassesList = new List(7, true);
126        constrain(this, itsRelatedClassesList,
127                     0, 6, 1,
128                         GridBagConstraints.REMAINDER,
129                         GridBagConstraints.BOTH,
130                         0, 0, 0);
131             GridBagConstraints.BOTH, GridBagConstraints.NORTHWEST, 1, 0, 1, 0,
132
133
134
135
136
137
138
139
140
141 /**

```

- 84 -

Tue Nov 12 07:44:48 1996 4

```
RelatedFilesPanel.java_1
142 * A FilenameFilter object than scans for file ending in .class
143 */
144 class ClassFileFilter implements FilenameFilter {
145     public boolean accept(File dir, String name) {
146         return (name.endsWith(".class"));
147     }
148 }
```

- 85 -

```

StringResources.java_1      Tue Nov 12 07:44:48 1996      1
1   /*
2    * $Header: /ccm/meitca/hsl/zonesagents/bootstrap/StringResources.java 7      11/11/96 5:13p BillP $
3    *
4    * Copyright 1996 Horizon Systems Laboratory.
5    * Mitsubishi Electric Information Technology Center America.
6    *
7    * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITR.
8    *
9    * DESCRIPTION
10   * The String Resources for the bootstrap package
11   *
12   * $Log: /ccm/meitca/hsl/zonesagents/bootstrap/StringResources.java $
13   *
14   *    7 11/11/96 5:13p BillP
15   *    Made correction to company name.
16   *
17   *    6 10/07/96 4:44p Walsh
18   *    Save name & location of last agent launched. Restart Wizard pointing
19   *    at same agent
20   *
21   *    5 9/15/96 5:11p Walsh
22   *    Fix spelling mistakes
23   *
24   *    4 9/09/96 5:59p Walsh
25   *    Make constants final
26   *
27   *    3 9/05/96 6:37p Walsh
28   *    Add some string constants associated with the command line launch
29   *    utility
30   *
31   *    2 9/04/96 12:26p Walsh
32   *    Add launch error message
33   *
34   *    1 8/29/96 5:21p Walsh
35   *    initial version
36   *
37   */
38   package com.meitca.hsl.zonesagents.bootstrap;
39
40   class StringResources {
41
42   // AgentLaunchWizard.java
43   public static String WIZARD_TITLE = "Agent Launch Wizard";
44   public static String WIZARD_APPNAME = "AgentLaunchWizard";
45   public static String PROP_DESC = "Properties file for zones Agent Launch Wizard";
46
47   // FindFilePanel.java

```

- 86 -

```

2
StringResources.java_1      Tue Nov 12 07:44:48 1996
1
48   public static final String LAUNCH_ERROR_TITLE = "Launch Error";
49   public static final String NO_FILENAME = "You must specify the filename of the .class file containing the
50   nt.";
51   public static final String ENDWITH_CLASS = "You must specify a file ending with .class .";
52   public static final String FF_TEXT1 = "This wizard will assist you in launching Agents";
53   public static final String FF_TEXT2 = "Please select the .class file containing the ";
54   public static final String FF_TEXT3 = "agent you would like to launch.";
55   public static final String FF_TEXT4 = "";
56   public static final String FF_FIELD_LABEL = "Class File: ";
57   public static final String FF_BROWSE_BUTTON = "Browse...";
58   public static final String BROWSE_DLG_TITLE = "Open Agent Class File";
59
//ItineraryPanel.java
60   public static final String DEST_COL_TITLE = "Destination";
61   public static final String ACTION_COL_TITLE = "Action";
62   public static final String ITIN_TEXT1 = "An agent's itinerary specifies where an agent travels to and";
63   public static final String ITIN_TEXT2 = "what operations the agent performs as it travels. Please ";
64   public static final String ITIN_TEXT3 = "specify the itinerary below. You should enter the host- ";
65   public static final String ITIN_TEXT4 = "names of the machines you would like your agent to travel ";
66   public static final String ITIN_TEXT5 = "to as well as the methods you would like called at each stop.";
67   public static final String ITIN_TEXT6 = "";
68
// RelatedFilePanel.java
69   public static final String RF_TEXT1 = "You may specify a list of classes which should be sent with";
70   public static final String RF_TEXT2 = "the Agent. These should be classes which the Agent is likely";
71   public static final String RF_TEXT3 = "to use while it is travelling.";
72   public static final String RF_TEXT4 = "";
73   public static final String RF_TEXT5 = "Please select any classes you would like sent with the agent.";
74   public static final String RF_TEXT6 = "";
75   public static final String RF_TEXT7 = "";
76
// BootStrap.java
77   public static final String USAGE = "Usage: bootstrap [-d hostname,method] [-f relatedfile.class]\n\t[-h] a
78   tfile.class";
79   public static final String ILLEGAL_OPTS = "bootstrap: illegal option -- ";
80   public static final String SYNTAX_ERROR = "\tThe related files specified by the -f option\n\tmust be valid .
81   class files and must end\n\twith .class";
82   public static final String ILLEGAL_AGENT = "bootstrap: Illegal agent file -- ";
83   public static final String NO_AGENT = "bootstrap: No agent specified";
84   public static final String HELP = "\tThe agent files must be valid Java class files\n\tand must end wi
85   va class files and must end\n\twith .class";
86   public static final String ILLEGAL_DEST = "bootstrap: Illegal destination specification -- ";
87   public static final String ILLEGAL_METHOD = "bootstrap: Illegal method name -- ";
88   public static final String NO_DEST = "bootstrap: You must specify at least destination";
89
// shared strings
90

```

- 87 -

StringResources.java\_1    Tue Nov 12 07:44:48 1996    3

```
91        public static final String FILE_NOTFOUND = "Could not find the file specified." ;
92        public static final String IOERROR = "An error occurred while trying to access the file specified.
93        public static final String BAD_CLASS_FORMAT = "The file specified did not contain a valid Java cla
94        public static final String NOT_AGENT = "The file specified did not contain a Java Agent." ;
95        public static final String NO_SKELTON = "Could not locate the Agent's skeleton." ;
96        public static final String LAUNCH_ERROR = "An error occurred while attempting to launch the Agent."
97
98 }
```

```

AgentPackage.java_1      Tue Nov 12 07:45:01 1996      1
1   /* $Header: /com/meitca/hsl/zonesagents/conduit/AgentPackage.java 18 11/11/96 5:15p Billp $
2
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center, America.
5   * All rights reserved.
6
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC USA.
8
9   * DESCRIPTION
10  * Agent transportation package object.
11  *
12  * $Log: /com/meitca/hsl/zonesagents/conduit/AgentPackage.java $
13
14  * 18 11/11/96 5:15p Billp
15  *     Made correction to company name.
16  *
17  * 17 10/28/96 6:31p Walsh
18  *     Add more debugging info.
19  *
20  * 16 10/24/96 5:15p Walsh
21  *     add some debugging information related to memory usage.
22  *
23  * 15 10/23/96 2:32p Walsh
24  *     add more debugging print lines
25  *
26  * 14 10/12/96 9:57a Walsh
27  *     fix compiler bug
28  *
29  * 13 10/10/96 4:03p Walsh
30  *     Add persistence support
31  *
32  * 12 9/30/96 7:56p Walsh
33  *     clean up javaee components
34  *
35  * 11 9/29/96 4:03p Walsh
36  *     Allow Destination methods to be specified by name
37  *     Give Agent access to AgentPackage information
38  *
39  * 10 9/11/96 12:24p Walsh
40  *     remove unneeded import of CollaboratorAgent
41  *
42  * 9 9/09/96 4:59p Walsh
43  *     call Agent prepareForTransport, completedTransport methods
44  *
45  * 8 8/23/96 3:58p Walsh
46  *     Add support for ad-hoc method invocation
47

```

- 89 -

```

AgentPackage.java_1      Tue Nov 12 07:45:01 1996      2

48   * 7   8/22/96 5:45p Walsh
49   * Move the custom marshalling of the agent into the AgentPackage class
50   *
51   * 6   8/19/96 5:26p Walsh
52   * Add support for remote class locating
53   *
54   * 5   8/09/96 4:35p Walsh
55   * Setting up file headers
56   */
57
58 package com.meteica.hsl.zonesagents.conduit;
59
60 import java.io.*;
61 import java.net.*;
62 import java.rmi.server.MarshalException;
63
64 import com.meteica.hsl.zonesagents.shared.Agent;
65
66 import com.meteica.hsl.util.*;
67
68 /**
69  * An AgentPackage contains an Agent, its MobileCodebase and
70  * its itinerary. The AgentPackage is passed between ConduitServers
71  * when the Agent travels.
72  * @see ConduitServerImpl
73  * @author Thomas Walsh
74  */
75
76 public class AgentPackage {
77     Agent           itsAgent;
78     MobileCodebase itsMobileCodebase;
79     Itinerary       itsItinerary;
80     String          itsHomeCodebaseURL;
81     int             itsCID;
82
83
84
85     /**
86      * Constructs an AgentPackage
87      * @param agent The Agent
88      * @param code The Agent's mobile codebase
89      * @param itinerary The Agent's itinerary
90      * @param homeCodebaseURL The URL of the Agent's codebase
91      */
92     public AgentPackage(Agent agent, MobileCodebase codebase,
93                         Itinerary itinerary, String homeCodebaseURL) {
94         Debug.println("conduit.memory", 3, "Agentpackage " + this + " created");
}

```

```

AgentPackage.java_1      Tue Nov 12 07:45:01 1996      3
95   itsAgent = agent;
96   itsMobileCodebase = codebase;
97   itsItinerary = itinerary;
98   itsHomeCodebaseURL = homeCodebaseURL;
99   itsOID = 0;
100
101
102  protected void finalize() throws Throwable {
103      Debug.println("ConduitMemory", 3, "AgentPackage " + this + " finalized");
104      super.finalize();
105  }
106
107  /** Retrieves the Agent from the package */
108  public Agent getAgent() {
109      return itsAgent;
110  }
111
112  /** Retrieves the Agent's Mobile Codebase from the package */
113  public MobileCodebase getMobileCodebase() {
114      return itsMobileCodebase;
115  }
116
117  /** Retrieves the Agent's itinerary from the package */
118  public Itinerary getItinerary() {
119      return itsItinerary;
120  }
121
122  /** Retrieves the Agent's codebase URL from the package */
123  public String getHomeCodebaseURL() {
124      return itsHomeCodebaseURL;
125  }
126
127  /**
128   * Prepares the Agent package for network transmission. This method
129   * performs the conversion of the Agent to a network transmission
130   * format. This method also calls the Agent's prepareForTransport
131   * method.
132   */
133  public void prepareForTransport() {
134      Debug.println("Conduit", 2, "AgentPackage.prepareForTransport called");
135
136      // Give the agent a chance to do any work in preparation
137      // for transport
138      itsAgent.prepareForTransport();
139
140
141  */

```

AgentPackage.java\_1 Tue Nov 12 07:45:01 1996 4

```

142      * Restores the Agent package for network transmission from its
143      * network transmission format. This method also calls the Agent's
144      * completedTransport method.
145      */
146  public void restoreFromTransportFormat() {
147      Debug.println("conduit", 2, "AgentPackage.restoreFromTransportFormat called");
148
149      /* give the agent a chance to do any work in preparation
150      // for arrival at destination
151      itsAgent.completedTransport();
152  }
153
154  private void readObject(ObjectInputStream stream)
155      throws IOException, ClassNotFoundException {
156      Debug.println("conduit", 2, "AgentPackage.readObject called");
157
158      itsMobileCodebase = (MobileCodebase)stream.readObject();
159      Debug.println("conduit", 4, "read MobileCodebase: " + itsMobileCodebase);
160
161      itsItinerary = (Itinerary)stream.readObject();
162      Debug.println("conduit", 4, "read Itinerary: " + itsItinerary);
163
164      itsHomeCodebaseURL = (String)stream.readObject();
165      Debug.println("conduit", 4, "read Codebase URL: " + itsHomeCodebaseURL);
166
167      // do the special deserialization of the Agent using the
168      // ConduitObjectInputStream class
169      ConduitServerClassLoader classLoader = new ConduitServerClassLoader(itsMobileCodebase, itsHomeCodebaseURL);
170
171      // unmarshal the agent, build the Agentpackage and then pass it along to receiveAgent
172      byte[] serializedAgent = (byte[])stream.readObject();
173      ByteArrayInputStream stream
174      ConduitObjectInputStream marshalinInput = new ConduitObjectInputStream(serializedAgent);
175
176      Debug.println("conduit", 4, "about to read Agent");
177      itsAgent = (Agent)marshalinInput.readObject();
178      Debug.println("conduit", 4, "read Agent: " + itsAgent);
179
180      // Restore the Agents reference to its Package will be
181      // restored when the agent arrives at the Destination
182      itsAgent.setPackage(this);
183
184  private void writeObject(ObjectOutputStream stream)
185      throws IOException {

```

AgentPackage.java\_1

Tue Nov 12 07:45:01 1996

5

```

187
188 stream.writeObject(itsMobileCodebase);
189 stream.writeObject(itsItinerary);
190 stream.writeObject(itsHomeCodebaseURL);
191
192 // do the special deserialization of the Agent using the
193 // ConduitObjectOutputStream class
194
195 // null out the Agents pointer to the AgentPackage.  We have to
196 // do this because Java object persistence does not support
197 // isomorphic persistence.  Since the AgentPackage has a
198 // reference to the Agent and the Agent has a reference
199 // to the AgentPackage we have a circular reference.
200 // Without isomorphic persistence, this circular reference
201 // would cause the object serialization code to gc into an
202 // infinite loop.  The reference to itsPackage will be
203 // restored when the agent arrives at the Destination
204 itsAgent.setPackage(null);
205
206 BytearrayOutputStream byteOutputStream = new BytearrayOutputStream();
207 ConduitObjectOutputStream marshallOutput = new ConduitObjectOutputStream(byteOutputStream);
208
209 // use object serialization to stream the agent to a byte array
210 marshallOutput.writeObject(itsAgent);
211 byte[] serializedAgent = byteOutputStream.toByteArray();
212
213 // finally write the serialized agent to the original output stream
214 stream.writeObject(serializedAgent);
215 }
216
217 /**
218 * Retrieves the AgentSkeleton object corresponding to the Agent.  */
219 AgentSkeleton getAgentSkeleton() throws ClassNotFoundException, InstantiationException, IllegalAccessException;
220 {
221     String agentClassname = itsAgent.getClass().getClassName();
222     String skelClassname = agentClassname + "_Skel";
223     ConduitServerClassLoader classLoader = (ConduitServerClassLoader) itsAgent.getAgent().getClass().getClassLoader();
224     if (classLoader != null)
225         return (AgentSkeleton) classLoader.loadClass(skelClassname).newInstance();
226     else
227         return (AgentSkeleton) Class.forName(skelClassname).newInstance();
228
229     void setOID(int oid) {
230         itsOID = oid;
231     }

```

- 93 -

AgentPackage.java\_1        Tue Nov 12 07:45:01 1996        6

```
232
233     int getOID() {
234         return intOID;
235     }
236 }
```

- 95 -

```

AgentSkeleton.java_1      Tue Nov 12 07:45:01 1996      1

1  /* $Header: /com/meitca/hel/zncesjagents/conduit/AgentSkeleton.java 3   11/11/96 5:15p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center, America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC IPA.
8  */
9  /*
10 * DESCRIPTION
11 * Agent method invocation skeleton interface
12 * Slogon: /com/meitca/hel/zncesjagents/conduit/AgentSkeleton.java $ 
13 *          3 11/11/96 5:15p Billp
14 *          * Made correction to company name.
15 *          * 2 8/29/96 5:15p Walsh
16 *          * Add getMethods method
17 *          * 1 8/23/96 2:54p Walsh
18 *          * initial versions
19 *          */
20 package com.meitca.hel.zncesjagents.conduit;
21
22 import com.meitca.hel.zncesjagents.shared.Agent;
23
24 /**
25  * import com.meitca.hel.zncesjagents.shared.Agent;
26 */
27 /**
28  * An interface representing an Agent's ability to invoke methods
29  * on the skeleton allowing the ConduitServer to make a call to any method within
30  * the Agent. At runtime, the ConduitServer looks at the Agent's itinerary
31  * and determines what method should be called at the current location. The
32  * skeleton allows the Conduit Server to call the method.
33  * @see
34  * @see ConduitServer
35  * @author
36  *        Thomas Walsh
37 public interface AgentSkeleton {
38 /**
39  * Invokes a method on an agent
40  * @param agent The agent whose method to invoke.
41  * @param i The method ID number of the method to invoke.
42  *          */
43  public void invoke(Agent agent, int i) throws IllegalAccessException;
44 /**
45  * Retrieves a list of the agent methods which can be invoked by
46  *          */
47

```

- 95 -

Agentskeleton.java\_1        Tue Nov 12 07:45:01 1996        2

```
48        * the skeleton.
49        */
50        public String[] getMethods() {
51        }
52 }
```

AgentThread.java\_1 Tue Nov 12 07:45:02 1996

```

1  /* $Header: /com/mitsc/hs1/zonesagents/conduit/AgentThread.java 4   11/11/96 5:15p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  *
9  * DESCRIPTION
10 * provides the thread in which an Agent will executes.
11 *
12 * $Log: /com/mitsc/hs1/zonesagents/conduit/AgentThread.java $
13 *
14 *        4 11/11/96 5:15p Billp
15 *        made correction to conduit's name.
16 *
17 *        3 10/29/96 5:29p Walsh
18 *        use errorLog class to report errors
19 *
20 *        2 10/28/96 6:21p Walsh
21 *        Add more debugging info
22 *
23 *        1 10/25/96 5:02p Walsh
24 *        Make AgentThread public
25 *
26 */
27 package com.mitsc.hs1.zonesagents.conduit;
28
29 import java.util.*;
30
31 import com.mitsc.hs1.zonesagents.shared.*;
32 import com.mitsc.hs1.zonesagents.conduit.*;
33
34 /**
35 * A subclass of Thread. AgentThread provides the thread in which an Agent will
36 * execute.
37 * @see Agent
38 * @see ConduitServer
39 * @author Thomas Walsh
40 */
41 public class AgentThread extends Thread {
42     AgentPackage itsPackage;
43
44 /**
45 * Constructs an AgentThread
46 * @param agentPackage The package containing the agent, its code and
47 * its itinerary.

```

```

AgentThread.java_1      Tue Nov 12 07:45:02 1996      2

48   */
49   public AgentThread( AgentPackage agentPackage) {
50     Debug.println("conduit.memory", 3, "AgentThread" + this + " created");
51     itsPackage = agentPackage;
52     this.start();
53   }

54   protected void finalize() throws Throwable {
55     Debug.println("conduit.memory", 3, "AgentThread" + this + " finalized");
56   }

57   /**
58    * Retrieves a reference to the Agent which is executing within this
59    * AgentThread
60    */
61   public Agent getAgent() {
62     return itsPackage.getAgent();
63   }

64   /**
65    * Retrieves a reference to the AgentPackage contained within this
66    * AgentThread
67   */
68   public AgentPackage getAgentPackage() {
69     return itsPackage;
70   }

71   /**
72    * Begins the Agent's execution
73    */
74   public void run() {
75   }

76   /**
77    * Begins the Agent's execution
78   */
79   public void main() {
80     try {
81       ConduitServerImpl.getPersistenceStore().addStore(itsPackage);
82     AgentSkeleton skel = itsPackage.getAgentskeleton();
83     Destination current = itsPackage.getItinerary().getCurrentLocation();
84     if (current.getMethodID() == Destination.UNKNOWN_METHOD_ID) {
85       current.convertNameToID(skel);
86     }
87     skel.invoke(itsPackage.getAgent(),
88     current.getMethodID());
89     itsPackage.getItinerary().getCurrentLocation().setCompleted();
90     ConduitServerImpl.getPersistenceStore().updateStore(itsPackage);
91     } catch (Exception error) {
92       ErrorLog.println("conduit.AgentThread");
93     }
94   }

```

```

AgentThread.java_1 Tue Nov 12 07:45:02 1996 3

95      ErrorLog.SEVERITY_ERROR,
96      "An error occurred while executing Agent :",
97      error);
98  }
99
100     ConduitServerImpl.decrementAgentCounters();
101
102     try {
103         // pass the agent along.
104         ConduitServerImpl.sendPackage(itsPackage);
105     } catch (NoSuchElementException done) {
106         Debug.println("conduit", 1, "AgentThread.run: No more destinations for Agent " +
107             itsPackage.getAgent().getId());
108     } catch (Exception error) {
109         ErrorLog.println("conduit.AgentThread",
110             ErrorLog.SEVERITY_ERROR,
111             "An error occurred while transporting Agent :",
112             error);
113     }
114
115     // remove the agent from the persistence store
116     ConduitServerImpl.getPersistenceStore().removeFromStore(itsPackage);
117
118 }

```

ConduitInputStream.java\_1 Tue Nov 12 07:45:02 1996 1

```

1  * $Header: /com/meitca/hsi/zonesagents/conduit/ConduitObjectInputStream.java 9 11/11/96 5:15p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC IVA.
8  *
9  * DESCRIPTION
10 * Specialized ObjectOutputStream used for agent deserialization
11 *
12 * $Log: /com/meitca/hsi/zonesagents/conduit/ConduitObjectInputStream.java $
13 *
14 * 9 11/11/96 5:15p Billp
15 * Made correction to company name.
16 *
17 * 8 11/01/96 5:32p Walsh
18 * Modify debugging levels of debug output
19 *
20 * 7 10/21/96 4:26p Walsh
21 * Add debug logging
22 *
23 * 6 9/30/96 7:05p Walsh
24 * clean up javadoc comments
25 *
26 * 5 9/11/96 6:40p Walsh
27 * fix compiler error
28 *
29 * 4 9/11/96 6:33p Walsh
30 * derive from MarshalStream
31 *
32 * 3 8/22/96 2:13p Walsh
33 * Clean up code a little
34 *
35 * 2 8/13/96 5:09p Walsh
36 * Added comments to documentation. resolveClass now uses
37 * ConduitServerClassLoader
38 *
39 * 1 8/12/96 7:17p Walsh
40 * Initial versions
41 */
42 package com.meitca.hsi.zonesagents.conduit;
43
44 import java.io.InputStream;
45 import java.io.IOException;
46 import java.io.StreamCorruptedException;
47 import java.lang.ClassNotFoundException;

```

- 19 -

```

ConduitobjectOutputStream.java_1 Tue Nov 12 07:45:02 1996 1
1 /* SHeader: /com/meitca/hsl/zonesagents/conduit/ConduitObjectOutputStream.java 7 11/11/96 5:15p Billp $
2 *
3 * Copyright 1996 Horizon Systems Laboratory,
4 * Mitsubishi Electric Information Technology Center America.
5 * All rights reserved.
6 *
7 * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8 *
9 * DESCRIPTION
10 * Specialized MarshalInputStream used for agent deserialization
11 *
12 * $Log: /com/meitca/hsl/zonesagents/conduit/ConduitObjectOutputStream.java $
13 *
14 *    7 11/11/96 5:15p Billp
15 *      Made correction to company name.
16 *
17 *    6 11/01/96 5:32p Walsh
18 *      Modify debugging levels of debug output
19 *
20 *    5 10/21/96 4:26p Walsh
21 *      Add debug logging
22 *
23 *    4 9/30/96 7:06p Walsh
24 *      clean up javadoc comments
25 *
26 *    3 9/11/96 6:33p Walsh
27 *      derive from MarshalStream
28 *
29 *    2 8/22/96 2:19p Walsh
30 *      Clean up code a little
31 *
32 *    1 8/12/96 7:17p Walsh
33 *      Initial versions
34 */
35 package com.meitca.hsl.zonesagents.conduit;
36
37 import java.io.OutputStream;
38 import java.io.IOException;
39 import java.io.StreamCorruptedException;
40 import java.lang.ClassNotFoundException;
41 import java.rmi.server.MarshalOutputStream;
42
43 import com.meitca.hsl.util.*;
44
45 public class ConduitObjectOutputStream
46     extends MarshalOutputStream {

```

```
ConduitObjectInputStream.java_1           Tue Nov 12 07:45:02 1996      3
95                                         Debug.println("conduit.classload", 3, "ConduitObj
96                                         );
97                                         return itsClassLoader.loadClass(classname);
98                                         }
99                                         }
```

- 102 -

```

ConduitObjectInputStream.java_1      Tue Nov 12 07:45:02 1996      2

48 import java.rmi.server.MarshalInputStream;
49 import java.net.MalformedURLException;
50 import java.net.URL;
51 import java.rmi.server.StubClassLoader;
52
53 import com.meitca.util.*;
```

54

55 */\**

56 \* A subclass of ObjectInputStream which is used to handle deserialization
 \* of Agent objects. ConduitObjectInputStream overrides the
 \* ObjectInputStream.resolveClass method. As the deserialization process
 \* encounters object contained within the agent, resolveClass is called for
 \* each unique class. resolveClass makes use of a ConduitServerClassLoader
 \* object to search the agents MobileCodebase for class bytecodes.

57 \* @see Agent

58 \* @see ConduitServerClassLoader

59 \* @see ObjectInputStream

60 \* @see MobileCodebase

61 \* @author Thomas Walsh

62 \*/

63 class ConduitObjectInputStream
 extends MarshallInputStream {

64 */\** Class loader object used for deserializing an agent. \*/
 65 ConduitServerClassLoader itsClassLoader;

66 }

67 */\**

68 \* Constructs a ConduitObjectInputStream.
 \* @param is the InputStream containing the Agent object.
 \* @param classLoader A ConduitServerClassLoader capable of loading
 \* classes from the Agent's MobileCodebase
 \*/

69 public ConduitObjectInputStream(InputStream is, ConduitServerClassLoader classLoader)
 throws IOException, StreamCorruptedException {

70 super(is);

71 itsClassLoader = classLoader;

72 }

73 */\**

74 \* Resolves a fully qualified class name to a Class object. resolveClass uses a
 \* ConduitServerClassLoader to actually perform the class loading.
 \* @param className The classes fully qualified class name.
 \*/

75 protected Class resolveClass(String className)

76 throws IOException, ClassNotFoundException {

77 URL url = new URL(className);

78 URLConnection connection = url.openConnection();

79 HttpURLConnection httpConnection = (HttpURLConnection)connection;

80 httpConnection.setRequestMethod("GET");

81 httpConnection.setDoOutput(true);

82 httpConnection.connect();

83 BufferedReader reader = new BufferedReader(new InputStreamReader(

84 httpConnection.getInputStream()));

85 String line;

86 while ((line = reader.readLine()) != null) {

87 if (line.startsWith("Content-Type: text/html; charset="))

88 continue;

89 else

90 return Class.forName(className);

91 }

92 throw new ClassNotFoundException(className);

93 }

94 }

-103-

```
ConduitObjectOutputStream.java_1      Tue Nov 12 07:45:02 1996    2
48     public ConduitObjectOutputStream(OutputStream is)
49         throws IOException {
50     super(is);
51 }
52
53     protected void annotateClass(Class c) throws IOException
54     Debug.println("conduit.classload", 2, "ConduitObjje
55     )
56
57 }
```

- 104 -

```

ConduitServer.java_1      Tue Nov 12 07:45:02 1996      1
1  /* $Header: /com/mexitca/hsl/zonesagents/conduit/ConduitServer.java 9      11/11/96 5:15p Billip $ */
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  *
9  * DESCRIPTION
10 * Conduit Server distributed interface
11 *
12 * $Log: /com/mexitca/hsl/zonesagents/conduit/ConduitServer.java $
13 *
14 * 9 11/11/96 5:15p Billip
15 *     Made correction to company name.
16 *
17 * 8 9/30/96 7:05p Walsh
18 *     clean up javadoc comments
19 *
20 * 7 8/27/96 3:51p Walsh
21 *     remove getConduitName method from ConduitServer interface
22 *
23 * 6 8/22/96 5:45p Walsh
24 *     move the custom marshalling of the agent into the AgentPackage class
25 *
26 * 5 8/19/96 5:26p Walsh
27 *     Add support for remote class loading
28 *
29 * 4 8/13/96 11:17a Walsh
30 *     add documentation & comments. Change signature of receivePackage
31 *     method.
32 *
33 * 3 8/12/96 7:15p Walsh
34 *     ConduitServer now performs Marshalling
35 *
36 * 2 8/09/96 4:26p Walsh
37 *     Setting up file headers
38 */
39
40 package com.mexitca.hsl.zonesagents.conduit;
41 import java.rmi.Remote;
42 import java.rmi.RemoteException;
43
44 import com.mexitca.hsl.zonesagents.conduit.AgentPackage;
45 import com.mexitca.hsl.zonesagents.conduit.MobileDatabase;
46
47

```

```

48   /**
49    * ConduitServer is an RMI distributed interface. It
50    * that allows conduit servers to communicate. Conduits
51    * by the ConduitServerImpl class. Conduit servers provide
52    * and execution
53    * and execution
54    * @see ConduitServerImpl
55    * @author Thomas Walther
56    */
57 public interface ConduitServer extends Remote {
58
59 /**
60  * Receive an agent from the previous conduit server
61  * @param a package The package containing
62  * @exception RemoteException if a communication
63  */
64
65 public void receivePackage(AgentPackage apackage)
66 throws

```

ConduitServerClassLoader.java\_1 Tue Nov 12 07:45:02 1996

```

1  /* $Header: /ccm/meitca/hsl/zonesjagents/conduit/ConduitServerClassLoader.java 12 11/11/96 5:15p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory.
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * Conduit Server ClassLoader subclass
11 *
12 * $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServerClassLoader.java $
13 *
14 * 12 11/11/96 5:15p Billp
15 * Made correction to company name.
16 *
17 * 11 11/01/96 5:32p Walsh
18 * Modify debugging levels of debug output
19 *
20 * 10 10/24/96 5:15p Walsh
21 * add some debugging information related to memory usage.
22 *
23 * 9 10/21/96 4:26p Walsh
24 * Add debug logging
25 *
26 * 8 9/30/96 7:06p Walsh
27 * clean up javadoc comments
28 *
29 * 7 9/28/96 6:37p Walsh
30 * use the AgentCodebase class to retrieve bytecodes
31 *
32 * 6 8/22/96 5:42p Walsh
33 * Cache Classes we define
34 *
35 * 5 8/19/96 5:25p Walsh
36 *
37 * 4 8/13/96 5:09p Walsh
38 * Added comments & documentation. loadClass now looks in MobileCodebase.
39 *
40 * 3 8/13/96 11:19a Walsh
41 * remove obsolete loadMobileClasses method
42 *
43 * 2 8/09/96 4:38p Walsh
44 * Setting up file headers
45 */
46 package com.meitca.hsl.zonesjagents.conduit;

```

- 107 -

ConduitServerClassLoader.java\_1 Tue Nov 12 07:45:02 1996 2

```

48 import java.rmi.Naming;
49 import java.io.*;
50 import java.net.*;
51 import java.util.*;
52
53
54 import com.meitaca.hsl.zonesagents.shared.*;
55 import com.meitaca.hsl.util.*;
56
57 /**
58 * A specialized ClassLoader object used by the conduit server to manage agent
59 * creation, deserialization, and sub-object creation. ConduitServerClassLoader
60 * extends the base system class loading behavior by allowing for classes to be
61 * loaded from the MobileCodebase of an Agent. For example, an Agent object might
62 * be created on one particular server on a network. When it is created, its
63 * bytecodes are added to the MobileCodebase. The agent now travels to a new
64 * server. The bytecodes for this particular agent have not been installed on the
65 * new server. Under normal circumstances, when the object deserializes code
66 * attempts to reconstruct the agent, a ClassCastException would result since
67 * the agents code does not exist anywhere on the new server.
68 * ConduitServerClassLoader allows the Java virtual machine to look for an Agents
69 * code in the MobileCodebase which allows Agent object to travel and pull their
70 * code with them.
71 * @see Agent
72 * @see ConduitServer
73 * @see MobileCodebase
74 * @see ClassReader
75 * @author Thomas Balish
76 */
77 public class ConduitServerClassLoader
78     extends ClassLoader {
79
80     /**
81      * The MobileCodebase corresponding to the Agent being managed. */
82      MobileCodebase mCodebase;
83
84     /**
85      * An URL back to the codebase of this agent on its home machine */
86
87     /**
88      * A hashtable containing the Class objects we have already
89      * defined during the serialization process
90      */
91     Hashtable itsPreviouslyDefinedClasses;
92
93     /**
94      * Constructs a ConduitServerClassLoader
95      * @param codebase The MobileCodebase corresponding to the

```

- 108 -

```

ConduitServerClassLoader.java_1 Tue Nov 12 07:45:02 1996 3

95      *          Agent being managed.
96      *          homeCodebaseURL An URL back to the codebase of this
97      *          agent on its home machine.
98      *          exception MalformedURLException if the URL passed in as the
99      *          homeCodebaseURL parameter is invalid.
100     */
101    public ConduitServerClassLoader(URLConnection codebase, String homeCodebaseURL)
102        throws MalformedURLException {
103        Debug.println("conduit.memory", 3, "Creating ConduitClassLoader " + this + " created");
104        itsMobileCodebase = codebase;
105        itsHomeCodebase = new AgentCodebase(homeCodebaseURL);
106        itsPreviousDefinedClasses = new HashSet();
107    }

108    protected void finalize() throws Throwable {
109        Debug.println("conduit.memory", 3, "ConduitServerClassLoader " + this + " finalized");
110        super.finalize();
111    }

112    /**
113     *          Loads a class. First checks on the local system and then looks in the
114     *          MobileCodebase.
115     *          param name Fully qualified name of the class to be loaded (ie <b>
116     *          java.lang.String</b>).
117     *          param resolve boolean value indicating whether or not the class
118     *          should be resolved via the ClassLoader.resolveClass method
119     */
120    public Class loadClass(String name)
121        throws ClassNotFoundException {
122        return loadClass(name, true);
123    }

124    /**
125     *          Loads a class, inherited from ClassLoader. First check on the local
126     *          system and then looks in the MobileCodebase.
127     *          param name Fully qualified name of the class to be loaded (ie
128     *          java.lang.String).
129     *          param resolve boolean value indicating whether or not the class
130     *          should be resolved via the ClassLoader.resolveClass method
131     */
132    protected Class loadClass(String name, boolean resolve)
133        throws ClassNotFoundException {
134        Debug.println("conduit.classload", 2, "ConduitServerClassLoader.loadClass called for " + nam
135        Debug.println("conduit.classload", 2, "resolve is " + resolve);
136        // first check the cache to see if we have encountered this class
137        // before. Note: A ClassLoader object should always do something
138
139
140
141

```

```

ConduitServerClassLoader.java_1 Tue Nov 12 07:45:02 1996 4

142   /* like this. In order for Java's type checking to work properly,
143   * every time a icaclass is called for a particular class name, the
144   * exact same Class object must be returned.
145   */
146   Class cache = (Class)itsPreviouslyDefinedClasses.get(name);
147
148   if (cache == null) {
149     System.out.println("conduit.classload", 3, "Class found in cache");
150
151   }
152
153   try {
154     /* first check this system
155     Class c = findSystemClass(name);
156     Debug.printin("conduit.classload", 3, "Class found on system");
157
158     if (c != null) {
159       catch (ClassNotFoundException e) {
160         /* it wasn't found on this system. Check the MobileCodebase...
161         bytecodes = itsMobileCodebase.genCode(name);
162
163         if (bytecodes != null) {
164           Debug.printin("conduit.classload", 3, "Class found in MobileCodebase");
165
166           /* bytecodes for the class were found in the MobileCodebase
167           /* try to define a class from them.
168           Class c = defineClassFromBytes(0, bytecodes.length);
169           if (c != null) {
170             Debug.printin("conduit.classload", 3, "Class defined from bytes");
171
172             /* store this class in the cache
173             itsPreviouslyDefinedClasses.put(name, c);
174
175             return c;
176         } else
177           Debug.printin("conduit.classload", 3, "Looking in home directory");
178           return loadClassFromHomeDatabase(name, recursive);
179
180       }
181
182
183
184
185   * Loads a class using the remote class loader.
186   * @param name fully qualified name of the class to be loaded (ie
187   * java.lang.String).
188   * @param recursive boolean value indicating whether or not the class

```

ConduitServerClassLoader.java\_1 Tue Nov 12 07:45:02 1996 5

```
189      *
190      */
191  protected Class loadClassFromHomeCodebase(String name, boolean resolve)
192  throws ClassNotFoundException {
193
194  " + name);
195  Debug.println("conduit.classload", 2, "ConduitServerClassLoader loadClassFromHomeCodebase called f..");
196  Debug.println("conduit.classload", 2, "resolve is " + resolve);
197  byte[] bytecodes = itsHomeCodebase.retrieveCode(name);
198
199  // bytecodes for the class were found in at remote location.
200  // store them in the MobileCodebase for future reference
201  itsMobileCodebase.storeCode(name, bytecodes);
202
203  // Try to define a class from them.
204  Class c = defineClass(bytecodes, 0, bytecodes.length);
205  if (resolve) {
206    resolveClass(c);
207
208    // store this class in the cache
209    itsPreviouslyDefinedClasses.put(name, c);
210
211  }
212  return c;
213
214 }
```

-110 -

- AAA -

```

ConduitServerImpl.java_1      Tue Nov 12 07:45:02 1996      1
1  /* $Header: /com/meicca/hsl/zonesjagents/conduit/ConduitServerImpl.java 23   11/11/96 5:15p Billip $
2  *
3  * Copyright 1996 Horizon Systems Laboratory.
4  * Mitsubishi Electric Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITCA.
8  */
9  * DESCRIPTION
10 * Conduit Server Implementation
11 *
12 * $Log: com.maicca.hsl/zonesjagents/conduit/ConduitServerImpl.java $
13 *
14 *    23 11/11/96 5:15p Billip
15 *        made correction to company name.
16 *
17 *    22 10/30/96 2:52p Walsh
18 *        Allow Debug and ErrorLog settings to be set in properties file
19 *
20 *    21 10/29/96 5:29p Walsh
21 *        use ErrorLog class to report errors
22 *
23 *    20 10/26/96 6:31p Walsh
24 *        Add more debugging info
25 *
26 *    19 10/25/96 5:03p Walsh
27 *        Make AgentThreaded public.  Provide no own source file
28 *
29 *    18 10/24/96 5:03p Walsh
30 *        add some debugging info related to memory usage.
31 *
32 *    17 10/23/96 2:12p Walsh
33 *        add more Debugging info
34 *
35 *    16 10/21/96 4:27p Walsh
36 *        Add debug logging
37 *
38 *    15 10/11/96 2:20p Walsh
39 *        remove restarted agents from persistence store
40 *
41 *    14 10/10/96 6:09p Walsh
42 *        Add persistence support
43 *
44 *    13 9/29/96 4:03p Walsh
45 *        Allow Destination methods to be specified by name.
46 *        Give Agent access to AgentPackage information
47

```

- 112 -

```

ConduitServerImpl.java_1      Tue Nov 12 07:45:02 1996      2

48   * 12  9/09/96 6:02p Walsh
49   * make constant final
50   *
51   * 11  9/04/96 12:32p Walsh
52   * Centralize manipulation of ConduitServer's RMI proxy.
53   * Move test security manager out of this file.
54   *
55   * 16  8/30/96 2:53p Walsh
56   * null out itsPackage (in AgentThread.run) to try to hedge the garbage
57   * collector.
58   *
59   * 9  8/23/96 3:58p Walsh
60   * Add support for ad-hoc method invocation
61   *
62   * 8  8/22/96 6:23p Walsh
63   * Add static reference to the one ConduitServer that exists.
64   *
65   * 7  8/22/96 5:45p Walsh
66   * Move the custom marshalling of the agent into the AgentPackage class
67   *
68   * 6  8/19/96 5:26p Walsh
69   * Add support for remote class loading
70   *
71   * 5  8/13/96 5:12p Walsh
72   * Changed signature of receivePackage. Changed marshalling,
73   * un-marshalling code to marshal only Agent and not entire AgentPackage
74   *
75   * 4  8/13/96 10:35a Walsh
76   * Add some comments and documentation
77   *
78   * 3  8/12/96 7:12p Walsh
79   * ConduitServer now performs marshalling
80   *
81   * 2  8/09/96 4:38p Walsh
82   * Setting up fine borders
83   */
84 package com.mitcalhei.zonesagents.conduit;
85
86 import java.rmi.server.UnicastRemoteServer;
87 import java.rmi.RemoteException;
88 import java.rmi.Naming;
89 import java.rmi.NotBoundException;
90 import java.net.*;
91 import java.io.*;
92 import java.util.*;
93
94 import com.mitcalhei.zonesagents.shared.*;

```

- 123 -

```

ConduitServerImpl.java_1   Tue Nov 12 07:45:02 1996   3

95 import com.meitca.hsl.util.*;
96 import com.meitca.hsl.zonesagents.security.InsecurityManager;
97
98 /**
99  * An RMI distributed object which provides for agent travel and execution.
100  * ConduitServerImpl implements the ConduitServer interface. ConduitServer
101  * is an RMI distributed interface. ConduitServerImpl is responsible for
102  * receiving incoming agents, providing for execution of agents, and for
103  * transmitting agents to other destinations.
104  * @see Agent
105  * @see ConduitServer
106  * @author Thomas Weish
107 */
108 public class ConduitServerImpl {
109     extends UnicastRemoteServer
110     implements ConduitServer {
111
112     /**
113      * The conduit server is a sentinel object. There will only be one of
114      * them running within a given Java virtual machine. The static member
115      * theServer is a pointer to that one ConduitServerImpl object
116      */
117     static ConduitServerImpl theServer;
118
119     /** The conduit servers name. */
120     String itsName;
121
122     /** The conduit servers properties file. */
123     Properties itsProperties;
124
125     /**
126      * An object representing the servers persistence store */
127     ConduitServerPersistence itsPersistenceStore;
128
129     /**
130      * The name by which a conduit server is bound in the RMI Registry */
131     static final String RMI_NAME = "ConduitServer";
132     static final String PROP_DESCR = "properties File for Conduit Server";
133
134
135     /**
136      * The following member variables store some statistical information
137      * about the performance of the Conduit Server.
138      */
139     Date itsStartTime;
140
141     /**
142      * The number of agents currently in the system */

```

```

ConduitServerImpl.java_1      Tue Nov 12 07:45:02 1996   4

        long          itsCurrentAgentCount;
142
143
144    /**
145     * The total number of agents to travel through the system
146     * since starting.
147     */
148
149
150    /**
151     * The largest number of agent to be in the system at any one time.
152     */
153
154
155
156
157    /**
158     * Constructs a conduit server
159     * @param name The servers name.
160     * @exception RemoteException if an error occurs setting up network
161     * communication.
162     */
163
164 public ConduitServerImpl(String name) throws RemoteException {
165     itsName = new String(name);
166     theServer = this;
167
168     // initialize the counters
169     itsTotalAgentCount = 0;
170     itsCurrentAgentCount = 0;
171     itsPeakAgentCount = 0;
172
173     itsProperties = new JASProperties(FILE_NAME,
174
175     JASProperties.SERVER,
176     PROP_DESC,
177     false);
178
179     catch (Exception e) {
180         ErrorLog.println("ConduitServerImpl", ErrorLog.SEVERITY_WARNING,
181             "Could not access Properties file for Conduit Server");
182         ErrorLog.println("ConduitServerImpl", ErrorLog.SEVERITY_WARNING,
183             "Continuing with defaults.", e);
184
185     }
186
187     // Construct an empty properties object.
188     itsProperties = new JASProperties();
}

```

ConduitServerImpl.java\_1 Tue Nov 12 07:45:02 1996 5

```

189     Debug.enable(itsProperties);
190     ErrorLog.initialize(itsProperties);
191
192     itsPersistenceStore = new ConduitServerPersistenceStore();
193 }
194
195 /**
196 * Send the Agent on to the next conduit server in its Itinerary.
197 * @param      agentPackage The Agent Package containing the Agent.
198 * @exception  NoSuchElementException If the agent has completed its
199 *          Itinerary.
200 *
201 * @exception  RemoteException If an error occurs communicating with
202 *          the next server.
203 * @exception  NotBoundException If the server name indicated as the
204 *          next destination in the Itinerary cannot be found on the
205 *          network.
206 */
207 public static void sendPackage(AgentPackage agentPackage)
208     throws NoSuchElementException, IOException,
209            RemoteException, NotBoundException{
210
211     // retrieve a remote reference to the next destination
212     ConduitServer nextConduit = (ConduitServer)Naming.lookup(
213         buildRmiURL(agentPackage.getItinerary().nextDestination()
214         .getDestinationHost()));
215
216     agentPackage.prepareForTransport();
217
218     // pass the agent along
219     nextConduit.receivePackage(agentPackage);
220
221
222 /**
223 * Receive an Agent from the previous conduit server in its Itinerary.
224 * @param      agentData A stream of bytes containing the serialized
225 *          agent.
226 * @param      codebase The agents MobileCodebase
227 * @param      itinerary The agents' Itinerary
228 * @exception  RemoteException If an error occurs communicating with
229 *          the prior server.
230 */
231 public void receivePackage(AgentPackage agentPackage)
232     throws RemoteException {
233     Debug.println("conduit", 2, "ConduitServerImpl.receivePackage called");
234     agentPackage.restoreFromTransportForm();

```

116 -

```

ConduitServerImpl.java_1   Tue Nov 12 07:45:02 1996   6

235     Debug.println("conduit", 1, "Agent " + agentPackage.getAgent() .getAgentID() + " has arrived");
236
237     incrementAgentCounters();
238
239     AgentThread newThread = new AgentThread(agentPackage);
240
241     static public void restartAgent(AgentPackage agentPackage) {
242         Debug.println("conduit", 1, "Restarting Agent " + agentPackage.getAgent() .getAgentID());
243
244         incrementAgentCounters();
245
246     try {
247
248         if (agentPackage.getItinerary() .getCurrentLocation() .hasCompleted()) {
249             Debug.println("conduit", 1, "\tAgent had completed execution on this Conduit Server");
250
251             Debug.println("conduit", 1, "\tSending Agent to next Destination");
252
253             decrementAgentCounters();
254
255             sendPackage(agentPackage);
256
257             // Remove the agent from the persistence store
258             getPersistenceStore() .removeFromStore(agentPackage);
259
260         } else {
261             Debug.println("conduit", 1, "\tAgent had not completed execution on this Conduit Server");
262
263             Debug.println("conduit", 1, "\tRestarting execution");
264             AgentThread newThread = new AgentThread(agentPackage);
265
266             try {
267                 newThread.start();
268             }
269
270         }
271
272         * Retrieves a property from the Conduit Server
273         * @param key The name of the property to retrieve
274         * @param def A default value which will be returned if
275         *          the property cannot be found
276         * @return the property value (a string)
277
278         public static String getProperty(String key, String def) {
279             return theServer.itsProperties.getProperty(key, def);

```

- 117 -

```

7 ConduitServerImpl.java_1 Tue Nov 12 07:45:02 1996
280 )
281 }
282 /**
283 * Returns the conduit servers name.
284 * @exception RemoteException if any network h
285 */
286 public static String getConduitName() throws Remo
287 return theServer.itsName;
288 }
289
290 /**
291 * This method is used internally to build a full
292 * server on the given host.
293 */
294 protected static String buildRmiURL(String hostnam
295 return new String("rmi://" + hostname + "."
296 )
297
298 /**
299 * Returns a reference to the conduit servers pers
300 */
301 public static ConduitServerPersistenceStore getCon
302 return theServer.itsPersistenceStore;
303
304 }
305
306 /**
307 * Returns the start time of this conduit serv
308 */
309 public static Date getStartTime() {
310 return theServer.itsStartTime;
311 }
312
313 /**
314 * Returns the uptime of this conduit server (in m
315 */
316 public static long getUptime() {
317 return (System.currentTimeMillis() - theServer.itsS
318 }
319
320 /**
321 * Returns the total number of agents to pass throu
322 */
323 public static long getTotalAgentCount() {
324 return theServer.itsTotalAgentCount;
325

```

```

ConduitServerImpl.java_1      Tue Nov 12 07:45:02 1996      8

327
328      }
329      /**
330      * Returns the number of agents currently executing
331      * in this Conduit Server.
332      */
333      public static long getCurrentAgentCount() {
334          return theServer.itsCurrentAgentCount;
335      }
336
337      /**
338      * Returns the maximum number of agents to be executing
339      * at any one time in this Conduit Server.
340      */
341      public static long getPeakAgentCount() {
342          return theServer.itsPeakAgentCount;
343      }
344
345      /**
346      *
347      */
348      public static synchronized void incrementAgentCounters() {
349          if (theServer != null) {
350              theServer.itsTotalAgentCount++;
351              theServer.itsCurrentAgentCount++;
352              if (theServer.itsCurrentAgentCount > theServer.itsPeakAgentCount)
353                  theServer.itsPeakAgentCount = theServer.itsCurrentAgentCount;
354          }
355      }
356
357      /**
358      *
359      */
360      public static synchronized void decrementAgentCounters() {
361          if (theServer != null) {
362              theServer.itsCurrentAgentCount--;
363          }
364      }
365
366      /**
367      * Performs server startup tasks
368      */
369      void startup() {
370          itsPersistenceStore.restorePersistedObjects();
371
372          // Monitor memory if conduit.memory facility is set to
373

```

```

ConduitServerImpl.java_1      Tue Nov 12 07:45:02 1996      9

374      // debug level 4
375      if (Debug.enabled("conduit.memory.watch", 4)) {
376          MemoryWatch watch = new MemoryWatch();
377      }
378
379      itsStartTime = new Date();
380  }

381
382  /**
383   * A simple bootstrap which brings the conduit server into existence,
384   * registers it name with the RMI naming service and begins listening
385   * for Agent requests. The main routine allows ConduitServerImpl to
386   * be initiated from the command line via the java command.
387   */
388
389  public static void main(String args[]) {
390      // Create and install the security manager
391      System.setSecurityManager(new StubSecurityManager());
392      System.setSecurityManager(new InsecureSecurityManager());
393
394      try { // construct the server and register it with the RMI Naming service
395          Debug.println("conduit", 1, "Create an Conduit Server");
396          String url = buildRmiURL(inetAddress.getHostName());
397
398          ConduitServerImpl server = new ConduitServerImpl(url);
399
400          Debug.println("conduit", 1, "Bind it to name: " + RMI_NAME);
401          Naming.rebind(RMI_NAME, server);
402
403          server.startup();
404          Debug.println("conduit", 1, "Conduit Server ready.");
405      } catch (Exception e) {
406          ErrorLog.println("conduit.main", ErrorLog.SEVERITY_FATAL,
407                          "an exception occurred during startup: ", e);
408      }
409  }
410

```

- 2 -

```

1  /*
2   * $Header: /com/meitca/hsl/zonesjagents/conduit/ConduitServerPersistenceStore.java 6      1
3   * Copyright 1996 Horizon Systems Laboratory,          11/11/96 5:15p Billp $
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   * DESCRIPTION
10  * Conduit Server persistence Store
11  *
12  * $Log: /com/meitca/hsl/zonesjagents/conduit/ConduitServerPersistenceStore.java $
13  *
14  *    6 11/11/96 5:15p Billp
15  *    Made correction to company name.
16  *
17  *    5 10/29/96 5:29p Walsh
18  *    Use ErrorLog class to report errors
19  *
20  *    4 10/21/96 4:26p Walsh
21  *    Add debug logging
22  *
23  *    3 10/21/96 11:38a Walsh
24  *    mirror change in Persistence API
25  *
26  *    2 10/18/96 10:58a Walsh
27  *    synch with change to findObject API
28  *
29  *    1 10/10/96 6:01p Walsh
30  *    initial version
31  *
32  */
33 package com.meitca.hsl.zonesjagents.conduit;
34 import java.io.File;
35 import java.util.*;
36 import java.util.*;
37
38 import com.meitca.hsl.zonesjagents.persistence.*;
39 import com.meitca.hsl.zonesjagents.shared.*;
40 import com.meitca.hsl.util.*;
41
42 /**
43  * Conduit Server Persistence Store
44  * @see ConduitServerImpl
45  * @author Thomas Walsh
46  */
47 class ConduitServerPersistenceStore {

```

- 12 -

```

ConduitServerPersistenceStore.java_1      Tue Nov 12 07:45:02 1996      2

48   // String constants representing the property names of
49   // two properties of interest
50   private static final String ENABLE_PERSISTENCE = "conduit.persistence.Enable";
51   private static final String PERSISTENCE_FILE = "conduit.persistence.FileName";
52   private static final String DEFAULT_FILENAME = "persist.store";
53
54   /** The PersistentStoreManager object used to manager the persistence store */
55   PersistentStoreManager itsStore;
56
57   /**
58    * Constructs the PersistenceStore. Persistence will be enabled only
59    * if the conduit.Persistence.Enable Property is set to "true". The
60    * PersistenceStore will be stored in the file named in the
61    * conduit.Persistence.FileName Property. If this Property is not set,
62    * the store will be placed in the file "zones.home/Agent/persist.store"
63    * only
64   */
65   ConduitServerPersistenceStore() {
66     if (new Boolean(ConduitServerImpl.getProperty(ENABLE_PERSISTENCE, "false")).booleanValue()) {
67       Debug.println("conduit.persistence", 1, "Persistence enabled.");
68
69     String filename = ConduitServerImpl.getProperty(PERSISTENCE_FILE, "");
70     Debug.println("conduit.persistence", 3, "conduit.Persistence.FileName property set to " + filename);
71
72     if (filename.length() == 0) {
73       filename = AgentConstants.AGENT_DIR + DEFAULT_FILENAME;
74     }
75
76     try {
77       // construct the PersistentStoreManager
78       Debug.println("conduit.persistence", 2, "Persistence filename is " + filename);
79       itsStore = new PersistentStoreManager(filename);
80     } catch (PersistentStoreException e) {
81       ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
82                       "An error occurred constructing the persistence store.", e);
83     }
84     ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
85                     "Continuing without persistence.");
86
87     // null out the itsStore reference. We will continue
88     // executing with persistence disabled
89     itsStore = null;
90   }
91
92   // null out the itsStore reference. This will

```

```

  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
  
```

122

/\*
 \* disable persistence
 \* itsStore = null;
 \*/

 /\*\*
 \* Adds an AgentPackage to the persistence store.
 \* @param aPackage The package to add.
 \*/
 void addToStore(AgentPackage aPackage) {
 if (itsStore != null) {
 if (aPackage.getOID() == 0) {
 Debug.println("conduit.persistence", 3, "Adding agent to persistence store: " + aPc-
age);
 }
 try {
 int oid = itsStore.insertObject(aPackage);
 Debug.println("conduit.persistence", 4, "Agent's oid is: " + oid);
 aPackage.setOID(oid);
 } catch (PersistentStoreDuplicateObjectException e1) {
 try {
 Debug.println("conduit.persistence", 4, "Agent already in store trying to
update");
 int cid = itsStore.findObject(aPackage);
 if (oid != PersistentStoreManager.INVALID\_OID) {
 Debug.println("conduit.persistence", 4, "Agent found in sto-
re");
 itsStore.updateObject(oid, aPackage);
 aPackage.setOID(oid);
 } else {
 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY\_ER-
OR,
 "An error occurred adding agent " + aPackage+
" to the persistence store.");
 }
 } catch (PersistentStoreManagerException e2) {
 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY\_ER-
OR,
 "PersistentStoreManager.findObject returned :
" + e2.getMessage());
 }
 }
 }
}

void updateObject(int oid, AgentPackage aPackage) {
 if (itsStore != null) {
 itsStore.updateObject(oid, aPackage);
 }
}

Tue Nov 12 07:45:02 1996

4

```

NING,
132      "Continuing without persistence.");
133
134      // null out the itsStore reference. We will continue
135      // executing with persistence disabled
136      itsStore = null;
137
138  } catch (PersistentStoreException e2) {
139      ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
140      "An error occurred adding agent " + aPackage.
141      " to the persistence store.", e2);
142
143      ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
144      "Continuing without persistence.");
145
146      // null out the itsStore reference. We will continue
147      // executing with persistence disabled
148      itsStore = null;
149
150  } catch (PersistentStoreException e3) {
151      ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
152      "An error occurred adding agent " + aPackage.getAgent
153      () .getAgentID() +
154      " to the persistence store.", e3);
155
156      ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
157      "Continuing without persistence.");
158
159      // null out the itsStore reference. We will continue
160      // executing with persistence disabled
161      itsStore = null;
162
163  }
164
165
166  /**
167   * Updates an AgentPackage in the persistence store.
168   * @param aPackage The package to add.
169   */
170 void updateStore(AgentPackage aPackage) {
171     if (itsStore != null) {
172         Debug.println("conduit.persistence", 3, "Updating agent in persistence store: " + aPackage);
173         Debug.println("conduit.persistence", 3, "Agent's oid is : " + aPackage.getOID());
174
try {

```

- 123 -

```

Tue Nov 12 07:45:02 1996      5

    itsStore.updateObject(aPackage.getOID(), aPackage);
176
177     ) catch (PersistentStoreException e2) {
178         ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
179         "An error occurred updating agent " + aPackage.getAgent() + ".  

AgentID() +
180
181         ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
182         " in the persistence store.", e2);
183
184         ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
185         "Continuing without persistence.");
186
187         // null out the itsStore reference. We will continue
188         // executing with persistence disabled
189         itsStore = null;
190
191     }
192
193     /**
194      * Removes an AgentPackage to the persistence store.
195      * @param aPackage The package to remove.
196      */
197     void removeFromStore(AgentPackage aPackage) {
198
199         if (itsStore != null) {
200             Debug.println("conduit.Persistence", 3, "Removing agent from persistence store: " + aPackage.
201             Debug.println("conduit.Persistence", 3, "Agent's oid is : " + aPackage.getOID());
202
203             try {
204                 itsStore.deleteObject(aPackage.getOID());
205             } catch (PersistentStoreException e) {
206                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_ERROR,
207                 "An error occurred removing agent " + aPackage.getAgent() + ".  

AgentID() +
208
209                 ErrorLog.println("conduit.Persistence", ErrorLog.SEVERITY_WARNING,
210                 "Continuing without persistence.");
211
212         // null out the itsStore reference. We will continue
213         // executing with persistence disabled
214         itsStore = null;
215
216     }
217
218
219     /**

```



Destination.java\_1 Tue Nov 12 07:45:03 1996 1

```

1  /* $Header: /com/meitca/bs1/zonesagents/conduit/Destination.java 7      11/11/96 5:15p Billp $
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America
5  * All rights reserved.
6  */
7  */
8  */
9  */
10 */
11 */
12 */
13 */
14 */
15 */
16 */
17 */
18 */
19 */
20 */
21 */
22 */
23 */
24 */
25 */
26 */
27 */
28 */
29 */
30 */
31 */
32 */
33 */
34 */
35 */
36 */
37 */
38 */
39 */
40 */
41 */
42 */
43 */
44 */
45 */
46 */
47 */

CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
```

DESCRIPTION

An agents destination. Contains RMI URL of conduit server, and method id of method to invoke at that location

SLog: /com/meitca/bs1/zonesagents/conduit/Destination.java \$

7 11/11/96 5:15p Billp

15 11/11/96 5:15p Billp

16 Made correction to company name.

17 \*

18 \* 6 10/10/96 6:02p Walsh

19 \* Add persistence support

20 \*

21 \* 5 9/30/96 7:04p Walsh

22 \* clean up javadoc comments

23 \*

24 \* 9/29/96 4:00p Walsh

25 \* Allow Destination methods to be specified by name

26 \* Give Agent access to RMI package information

27 \*

28 \* 3 9/04/96 12:31p Walsh

29 \* Destination now holds only the hostname of the ConduitServer, not the

30 \* entire RMI URL

21 \*

32 \* 2 6/29/96 5:15p Billp

33 \* add default constructor and set\* methods

34 \*

35 \* 1 8/23/96 3:56p Walsh

36 \* Initial versions

37 \*/

38 package com.meitca.bs1.zonesagents.conduit;

39 \*

40 import com.meitca.bs1.zonesagents.conduit.InvalidMethodNameException;

41 \*/

42 \*/
43 \* An object representing a Destination in an Agents Itinerary. The
44 \* Destination is composed of two parts: <br>
45 \* <ol>
46 \* <li>the hostname of the conduit server in which to travel
47 \* <li>An identifier indicating the method of the agent to call

- 127 -

**Destination.java-1** Tue Nov 22 07:45:03 1996 2

Destination.java\_1

Tue Nov 12 07:45:03 1996

3

```

95      */
96      public Destination(String hostname, int method) {
97          itsHostName = hostname;
98          itsMethodID = method;
99          itsMethodName = null;
100         itsHasCompleted = false;
101     }
102
103     /**
104      * Construction an empty destination object
105     */
106     public Destination() {
107         itsHostName = new String("");
108         itsMethodID = UNKNOWN_METHOD_ID;
109         itsMethodName = null;
110         itsHasCompleted = false;
111     }
112
113     /**
114      * Retrieves the hostname of the conduit server in which to travel
115     */
116     public String getDestinationHost() {
117         return itsHostName;
118     }
119
120     /**
121      * Retrieves the method ID number of the method to invoke
122     */
123     public int getMethodID() {
124         return itsMethodID;
125     }
126
127     /**
128      * Sets the URL of the conduit server in which to travel
129     */
130     public void setDestinationHost(String hostname) {
131         itsHostName = hostname;
132     }
133
134     /**
135      * Sets the method ID number of the method to invoke
136     */
137     public void setMethodID(int id) {
138         itsMethodID = null;
139         itsMethodID = id;
140     }
141

```

- 129 -

```

Destination.java_1      Tue Nov 12 07:45:03 1996   4

142     /**
143      * Retrieves the method name of the method to invoke
144      */
145     public String getMethodName() {
146         return itsMethodName;
147     }
148
149     /**
150      * Makes a clone of the Destination object
151     */
152     public Destination cloneDestination() {
153         Destination clone = new Destination();
154         clone.itsHostName = itsHostName;
155         clone.itsMethodID = itsMethodID;
156         clone.itsMethodName = itsMethodName;
157         return clone;
158     }
159
160     /**
161      * Sets the internal state of the Destination to
162      * indicate that execution has been completed
163      */
164     void setCompleted() {
165         itsHasCompleted = true;
166     }
167
168     /**
169      * Checks the internal state of the Destination to
170      * see if execution has been completed.
171      */
172     boolean hasCompleted() {
173         return itsHasCompleted;
174     }
175
176     /**
177      * Used internally to convert a method name into a method id
178      * protected void convertNameToMethodID(AgentSkeleton skel)
179      * throws InvalidMethodNameException !
180
181     if (itsMethodName == null) {
182         throw new InvalidMethodNameException("No name specified for destination");
183     }
184
185     String[] methods = skel.getMethods();
186     for (int i=0; i<methods.length; i++) {
187         if (itsMethodName.equals(methods[i])) {
188             itsMethodID = i;
189             itsMethodName = null;

```

- 130 -

InvalidMethodNameException.java\_1

Tue Nov 12 07:45:03 1996

```

1  /*
2   *  $Header: /ccm/meitca/hsl/zonesjagents/conduit/InvalidMethodNameException.java 3
3   *  Copyright 1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *  All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   */
9   * DESCRIPTION
10  * Exception indicating that Method Name specified in a Destination
11  * is not a valid name
12  */
13  * $Log: /com/meitca/hsl/zonesjagents/conduit/InvalidMethodNameException.java $
14  *
15  * 3 11/11/96 5:15p BillP
16  * Made correction to company name.
17  */
18  * 2 9/30/96 7:06p Walsh
19  * clean up javadoc comments
20  */
21  * 9/29/96 4:17p Walsh
22  * initial version
23  */
24  */
25 package com.meitca.hsl.zonesjagents.conduit;
26 /**
27 /**
28  * An exception which indicates that the method name that was specified
29  * in a Destination was not the name of a method of the agent which
30  * is travelling.
31  * @see Agent
32  * @see AgentSkeleton
33  * @author Thomas Walsh
34  */
35 public class InvalidMethodNameException extends Exception {
36     public InvalidMethodNameException() {
37         super();
38     }
39
40     public InvalidMethodNameException(String s) {
41         super(s);
42     }
43 }
```

-13-

- 132 -

```

Itinerary.java_1 Tue Nov 12 07:45:03 1996 1
1 /* $Header: /com/meitca/bsl/zonesjagents/conduit/itinerary.java 5 11/11/96 5:15p Billp $
2 *
3 * Copyright 1996 Horizon Systems Laboratory.
4 * Mitsubishi Electric Information Technology Center America.
5 * All rights reserved.
6 */
7 * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8 *
9 * DESCRIPTION
10 * Agent itinerary conduit
11 *
12 * $Log: /com/meitca/bsl/zonesjagents/conduit/itinerary.java $
13 *
14 * 5 11/11/96 5:15p Billp
15 * Made correction to company name.
16 *
17 * 4 9/29/96 4:09p Walsh
18 * Allow Destination methods to be specified by name
19 * Give Agent access to Agent.Package information
20 *
21 * 3 8/23/96 3:53p Walsh
22 * Add support for a direct method invocation
23 *
24 * 2 8/09/96 4:33p Walsh
25 * Setting up file handles
26 */
27 package com.mitsubishi.itl.zonesjagents.conduit;
28
29 import java.util.*;
30
31 /**
32 * An object representing an Agent's Itinerary. The Itinerary
33 * indicated WHERE an agent should travel and indicated WHEN
34 * the Agent should do at each stop
35 * @see Agent
36 * @see ConduitServer
37 * @see Destination
38 * @author Thomas Walsh
39 */
40 public class Itinerary {
41     Vector itsDestinations;
42     int itsCurrentStop;
43
44     /**
45      * Constructs an Itinerary object
46      */
47 }
```

- 133 -

```

Itinerary.java_1      Tue Nov 12 07:45:03 1996      2

48     itsDestinations = new Vector();
49     itsCurrentStop = -1;
50   }
51
52   /**
53    * Adds a new destination to the end of the Agents itinerary.
54    * @param the new destination
55    */
56   public void addDestination(Destination destination) {
57     itsDestinations.addElement(destination);
58   }
59
60   /**
61    * Clears all destinations from the Agents itinerary.
62    */
63   public void clearItinerary() {
64     itsDestinations.removeAllElements();
65   }
66
67   /**
68    * Resets the Itinerary back to the initial Destination.
69    * This method could be used to force an Agent to restart
70    * its itinerary and revisit all destinations
71    */
72   public void resetItinerary() {
73     itsCurrentStop = -1;
74   }
75
76   /**
77    * Retrieves the destination object identifying the current
78    * location of the agent.
79    * Returns the agents current location
80    * @exception NoSuchElementException if the agent has not
81    * travelled.
82   */
83   public Destination getCurrentLocation() throws NoSuchElementException {
84     try {
85       return (Destination)itsDestinations.elementAt(itsCurrentStop);
86     } catch (ArrayIndexOutOfBoundsException) {
87       catch (OutOfBoundsException boundsException) {
88         throw new NoSuchElementException();
89       }
90     }
91   }
92   /**
93    * Retrieves the destination object identifying the current
94    * location of the agent

```

```

Itinerary.java_1      Tue Nov 12 07:45:03 1996      3

95     * @returns      The Agent's next destination
96     * @exception    NoSuchElementException If the Agent has not
97     *                 travelled.
98   */
99   public Destination getNextDestination() throws NoSuchElementException {
100     try {
101       return (Destination)itsDestinations.nextElement(itIsCurrentStop,i);
102     } catch (ArrayIndexOutOfBoundsException boundError) {
103       throw new NoSuchElementException();
104     }
105   }
106   /**
107   * Retrieves an enumeration of all of the Destinations in the
108   * Itinerary.
109   * @return      An enumeration of all of the Destinations in the
110   *                 itinerary.
111   */
112   public Enumeration destinations() {
113     return itsDestinations.elements();
114   }
115   /**
116   * Makes a clone of the itinerary. If the reset parameter is true
117   * then the new itinerary is reset to the first destination. If
118   * reset is false, then the new itinerary is set such that the next
119   * destination is the same as that of the original
120   * parameter. Reset is true if the new itinerary should be reset to
121   * the first destination of the original itinerary.
122   * @param      itinerary      The original itinerary
123   * @param      clone          A new itinerary()
124   * @param      reset          Boolean to indicate if the new itinerary()
125   *                           should be reset to the first destination
126   */
127   public Itinerary cloneItinerary(boolean reset) {
128     Enumeration enum = itsDestinations.elements();
129     while (enum.hasMoreElements()) {
130       Destination original = (Destination)enum.nextElement();
131       Destination clonedDest = (Destination)original.clone();
132       clonedDest.addDestination(cloneDest);
133     }
134   }
135   /**
136   * @param      clone          A new itinerary()
137   * @param      itsCurrentStop = itsCurrentStop;
138   */
139   return clone;
140 }
141

```

Itinerary.java\_1      Tue Nov 12 07:45:03 1996      4

```
/*
 * Retrieves the next destination to which an Agent should
 * travel and increments the Itinerary's internal state to
 * the next destination. This method is called internally
 * by the ConduitServer when it is preparing to transport
 * the Agent to its next destination.
 */
protected Destination nextDestination() throws NoSuchElementException {
    try {
        return (Destination)itinerary.nextElement();
    }
    catch (ArrayOutOfBoundsException boundary) {
        throw new NoSuchElementException();
    }
}
```

```

MemoryWatch.java_1      Tue Nov 12 07:45:03 1996      1
1  /* $Header: /com/meitaca/nl/zonesagents/conduit/MemoryWatch.java 3   11/11/96 5:15p Billip $ */
2  *
3  * Copyright 1996 Horizon Systems Laboratory,
4  * Mitsubishi Electric Information Technology Center America
5  * All rights reserved
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA
8  *
9  * DESCRIPTION
10 * Conduit Server implementation
11 *
12 * $Log: /com/meitaca/nl/zonesagents/conduit/MemoryWatch.java 5
13 *          *
14 * 3  11/11/96 5:15p Billip
15 *     Made correction to company name.
16 *
17 * 2  19/26/96 6:31p Walsh
18 *     Add more debugging info
19 * 1  10/24/96 5:12p Walsh
20 *     Initial Version
21 *
22 */
23 package com.meitaca.nl.zonesagents.conduit;
24
25 import java.util.*;
26
27 import com.meitaca.conduit.*;
28
29 /**
30 * The memory watch class provides an minimum priority thread which
31 * reports the total amount of memory allocated to the Java VM as
32 * well as the amount of free memory currently available.
33 *
34 * To enable the MemoryWatch, the Conduit Server should be started
35 * up with the following switch passed into java.exe
36 * -DDebug="conduit.memory.watch=4"
37 * Author: Thomas Walsh
38 *
39 */
40 class MemoryWatch extends Thread {
41
42     /** The number of milliseconds between polls of the VM */
43     static int pollingInterval = 500; // in millisec
44
45     /** Construct a Memorywatch object */
46     MemoryWatch() {
47         super("conduit.memory.watch", "Memory Watch thread starting...");
```

```

MemoryWatch.java_1      Tue Nov 12 07:45:03 1996      2

48          setPriority(Thread.MIN_PRIORITY);
49          start();
50      }
51
52      /** Performs memory polling */
53      public void run() {
54          while (true) {
55              Debug.println("conduit.memory.watch", 4, "");
56              Debug.println("conduit.memory.watch", 4, "Server up since: " + ConduitServerImpl.getStartTime());
57
58              long uptime = ConduitServerImpl.getUptime();
59              Date up_date = new Date(uptime);
60              Date bigBang = new Date(0);
61              Debug.println("conduit.memory.watch", 4, "Server uptime: " +
62                           (up_date.getYear() - bigBang.getYear()) + " years, " +
63                           (up_date.getMonth() - bigBang.getMonth()) + " months, " +
64                           (up_date.getDate() - bigBang.getDate()) + " days, " +
65                           (up_date.getHours() - bigBang.getHours()) + " hours, " +
66                           (up_date.getMinutes() - bigBang.getMinutes()) + " minutes, " +
67                           (up_date.getSeconds() - bigBang.getSeconds()) + " seconds.");
68
69              Debug.println("conduit.memory.watch", 4, "Total number of Agents:\t\t\t" + ConduitServerImpl
70              GetCurrentAgentCount());
71              Debug.println("conduit.memory.watch", 4, "Current number of Agents:\t\t\t" + ConduitServerImpl
72              GetPeakAgentCount());
73              Debug.println("conduit.memory.watch", 4, "Total memory allocated to system:\t\t\t" + Runtime.getRun
74              time().totalMemory());
75              Debug.println("conduit.memory.watch", 4, "Total free memory available:\t\t\t" + Runtime.getRun
76              time().freeMemory());
77              try {
78                  sleep(pollingInterval);
79              } catch (InterruptedException e) {
80                  Debug.println("conduit.memory.watch", 4, "Some thread interrupted the MemoryWatch thi
81              }
82
83      }

```

```

MobileCodebase.java_1      Tue Nov 12 07:45:03 1996      1

1  /*
2   *  SHeader: /com/meitca/hs1/zonesagents/conduit/MobileCodebase.java 5
3   *  Copyright 1996 Horizon Systems Laboratory,
4   *  Mitsubishi Electric Information Technology Center America.
5   *  All rights reserved.
6   */
7  /*
8   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
9   */
10  /*
11   *  Agent Mobile codebase object
12   */
13  /*
14   *  $Log: /com/meitca/hs1/zonesagents/conduit/MobileCodebase.java $
15   *  5 11/11/96 5:15p Billp
16   *  Made correction to company name.
17   *  4 9/30/96 7:06p Walsh
18   *  clean up javadoc comments
19   */
20  /*
21   *  3 8/13/96 11:18a Walsh
22   *  Add documentation & comments. Set up use of Hashtable for bytecodes
23   *  storage.
24   *  2 8/09/96 4:28p Walsh
25   *  Setting up file headers
26   */
27 package com.meitca.hs1.zonesagents.conduit;
28
29 import java.util.Hashtable;
30
31 /**
32  * Contains the bytecodes for the objects that travel with an Agent.
33  * @see Agent
34  * @author Thomas Walsh
35  */
36 /**
37 public class MobileCodebase {
38 /**
39  * Provides for easy storage and retrieval of bytecodes. Bytecodes
40  * are stored in a Hashtable and are accessible by fully qualified
41  * Class name (ie "java.lang.String").
42  */
43 Hashtable itsByteCodes;
44
45 /**
46  * Constructs a MobileCodebase object
47 */

```

```
MobileCodebase.java_1      Tue Nov 12 07:45:03 1996      2
48     public MobileCodebase() {
49         itsByteCodes = new Hashtable();
50     }
51
52     /**
53      * Stores an objects bytecodes into the MobileCodebase. use.
54      * @param name The fully qualified Class name (ie
55      *             <i><b>"java.lang.String"</b></i>).
56      * @param byteCodes The objects bytecodes.
57      */
58     public void storeCode(String name, byte[] code) {
59         itsByteCodes.put(name, code);
60     }
61
62     /**
63      * Retrieves an objects bytecodes from the MobileCodebase.
64      * @param name The fully qualified Class name (ie
65      *             <i><b>"java.lang.String"</b></i>).
66      */
67     public byte[] getCode(String name) {
68         return (byte[]) itsByteCodes.get(name);
69     }
70 }
```

```

RemoteClassLoader.java_1      Tue Nov 12 07:45:14 1996      1
1  /*
2   * $Header: /com/mitsubishi/zonesagents/remoteLoader/RemoteClassLoader.java 4
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8
9   * DESCRIPTION
10  * Remote class loader distributed interface.
11  *
12  * $Log: /com/mitsubishi/zonesagents/remoteLoader/RemoteClassLoader.java $
13  * Revision 1.1 Date 11/11/96 5:15p BillP
14  * Nadie correction to company name
15  *
16  * 3 9/28/96 5:39p Walsh
17  * Add new parameter to retrieveClass method
18  *
19  * 2 8/27/96 4:10p Walsh
20  * retrieveClass throws ClassNotFoundException (compiler names)
21  *
22  * 1 8/19/96 5:28p Walsh
23  * Initial version
24  *
25  */
26 package com.mitsubishi.zonesagents.remoteLoader;
27
28 import java.rmi.Remote;
29 import java.rmi.RemoteException;
30
31 /**
32  * RemoteClassLoader is an RMI distributed interface. It provides a
33  * mechanism by which conduit servers can retrieve bytecodes for classes
34  * that exist on machines other than the one on which the conduit
35  * server is loading.
36  * @see ConduitServer
37  * @author Thomas Walsh
38  *
39  */
40 public interface RemoteClassLoader extends Remote {
41
42  /**
43   * Retrieve the bytecodes for the specified class.
44   * @param classURL An URL specifying the class to be loaded.
45   * @return The bytecodes for the class.
46   * @exception RemoteException If an error occurs communicating.
47

```

```
RemoteClassLoader.java_1           Tue Nov 12 07:45:14 1996    2
48      *
49      public byte[] retrieveClass(String codebaseURL, String
50
51 }
```

1721

```
RemoteClassLoaderImpl.java_1      Tue Nov 12 07:45:14 1996      1
1   /* $Header: /com/meitca/hsl/zones/jagents/remoteloader/RemoteClassLoader.java 8 11/11/96 5:15p Billip $
2
3   * Copyright 1996 Horizon Systems Laboratory,
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8
9   * DESCRIPTION
10  * Remote class loader implementation
11  *
12  * $Log: /com/meitca/hsl/zones/jagents/remoteloader/RemoteClassLoader.java $
13  *
14  *     8 11/11/96 5:15p Billip
15  *         Made correction to company name
16  *
17  *     7 10/29/96 5:31p Walsh
18  *         Use ErrorLog class to report errors
19  *
20  *     6 10/21/96 1:25p Walsh
21  *         Add some debug output
22  *
23  *     5 9/28/96 6:40p Walsh
24  *         Use AgentDatabase class to load byte codes.
25  *         Add retrieveRemoteClass to encapsulate RMI code
26  *
27  *     4 9/04/96 12:33p Walsh
28  *         Centralize manipulation of RMI URL.  Use the first security manager
29  *
30  *     3 8/27/96 4:18p Walsh
31  *         Add silly constructor (because Sun compiler wants it)
32  *
33  *     2 8/23/96 3:59p Walsh
34  *         Add logging message
35  *
36  *     1 8/19/96 5:28p Walsh
37  *         Initial version.
38  */
39 package com.meitca.hsl.zones.jagents.remoteloader;
40
41 import java.rmi.Remote;
42 import java.rmi.RemoteException;
43 import java.rmi.server.UnicastRemoteServer;
44 import java.rmi.server.StubSecurityManager;
45 import java.rmi.Naming;
46 import java.io.*;
47 import java.lang.ClassNotFoundException;
```

- 153 -

```

48
49 import com.metea.hsl.zonesagents.security.*;
50 import com.metea.hsl.zonesagents.shared.*;
51 import com.metea.hsl.util.*;
52
53 /**
54 * An RMI distributed object which provides for remote retrieval of class
55 * bytecodes. RemoteClassLoaderImpl implements the RemoteClassLoader interface.
56 * @see RemoteClassLoader
57 * @author Thomas Walsh
58 */
59
60 public class RemoteClassLoaderImpl:
61     implements RemoteClassLoader {
62
63     /**
64      * The name to which a conduit server is bound in the RMI Registry */
65     static String RMICLASSNAME = "RemoteClassLoader";
66
67     /**
68      * object which actually handles the dirty work of retrieving .class
69      * files for a given class.
70      */
71     ClassFileLoader
72
73     public RemoteClassLoaderImpl() throws RemoteException {
74         ityClassFileLoader = new ClassFileLoader();
75     }
76
77
78     /**
79      * Retrieve the bytecodes for the specified class. This method is invoked
80      * by RMI. The calling program on the other side of the RMI connection
81      * usually invokes the RMICLASSNAME by calling the
82      * RemoteClassLoader.retrieveRemoteClass method below.
83      * @param className A URL specifying the class to be loaded.
84      * @return The bytecodes for the class.
85      * @exception RemoteException If an error occurs communicating
86      *          with the database.
87      */
88     public byte[] retrieveClass(String className)
89         throws RemoteException, ClassNotFoundException {
90         Debug.println("remoteLoader", 2, "Loading ");
91         className += ".class";
92         if(className.length() == 0) {
93             print(className);
94         }

```

- 14c -

```

RemoteClassLoaderImpl.java_1      Tue Nov 12 07:45:14 1996      3

95      return itsClassFileLoader.loadClassFromFile(classPath(className));
96
97  } else {
98      Debug.println("remoteLoader", 3, "loading from directory");
99      return itsClassFileLoader.loadClassFromDirectory(codebase_dir, className);
100 }

101 /**
102 * Retrieves the bytecodes for the specified class from the remote
103 * codebase specified by codebase_URL. This method is called internally
104 * by the Java agent system whenever an object wants to retrieve code
105 * from a RemoteClassLoader. This method hides the details of the RMI.
106 * @param codebase_URL An URL specifying the class to be loaded.
107 * @param className The class name of the form /directory_path/classname
108 * @param bytecode The fully qualified class name of the class whose
109 * bytecode should be retrieved.
110 * @return The bytecodes for the class.
111 * @exception RemoteException RemoteException if an error occurs communicating.
112
113 public static byte[] retrieveRemoteClass(String codebase_URL, String className)
114     throws ClassNotFoundException {
115
116     Debug.println("remoteLoader", 2, "retrieveRemoteClass: loading " +
117         className + " from " + codebase_URL);
118
119     int count = 0;
120     int index = 0;
121     while (count < 4) {
122         index = codebase_URL.indexOf('/', index);
123         index = codebase_URL.indexOf('/', index);
124         if (index == -1)
125             break;
126
127         count++;
128         index++;
129     }
130
131     String rmiURL;
132     String code_dir;
133
134     if (index != -1) {
135         rmiURL = codebase_URL.substring(0, index - 1);
136         code_dir = codebase_URL.substring(index, codebase_URL.length());
137     } else {
138         rmiURL = codebase_URL;
139         code_dir = null;
140     }
141

```

```

RemoteClassLoaderImpl.java_1      Tue Nov 12 07:45:14 1996      4
142 Debug.println("remoteLoader", 3, "remote object URL: " + rmiURL);
143 Debug.println("remoteLoader", 3, "remote code dir : " + code_dir);
144
145 {
146     RemoteClassLoader remote = (RemoteClassLoader)Naming.lookup(rmiURL);
147
148     return remote.retrieveClass(code_dir, className);
149 }
150
151     catch (Exception e) {
152         throw new ClassNotFoundException("Could not load " + className + e.getMessage());
153     }
154 }
155
156 /**
157 * This method is used internally to build a full RMI URL for a remote
158 * class loader on the given host.
159 */
160 public static String buildRmiURL(String hostname) {
161     return new String("rmi://" + hostname + "/" + RMI_NAME);
162 }
163
164 /**
165 * This method is used internally to build a full RMI URL for a remote
166 * codebase on the given host.
167 */
168 public static String buildCodebaseURL(String hostname, String codeDir) {
169     return new String(hostname + "/" + codeDir);
170 }
171
172 /**
173 * A simple bootstrap which brings the remote class loader into existence,
174 * registers its name with the RMI naming service and begins listening
175 * for requests. The main routine allows RemoteClassLoaderImpl to
176 * be initiated from the command line via the java command.
177 */
178 public static void main(String args[]) {
179     // Create and install the security manager
180     System.setSecurityManager(new InsecurityManager());
181
182     try {
183         // Construct the server and register it with the RMI Naming service
184         Debug.println("remoteLoader", 1, "Create an RemoteClassLoaderImpl");
185         RemoteClassLoaderImpl loader = new RemoteClassLoaderImpl();
186
187         Debug.println("remoteLoader", 1, "Bind it to name: " + RMI_NAME);
188         Naming.rebind(RMI_NAME, loader);
189     }
190 }

```

RemoteClassLoader.java\_1        Tue Nov 12 07:45:14 1996        5

```
189              Debug.println("remoteLoader", 1, "RemoteClassLoaderImpl ready.");
190          } catch (Exception e) {
191              ErrorLog.println("remoteLoader.main", ErrorLog.SEVERITY_FATAL,
192                             "an exception occurred during startup:", e);
193          }
194          }
195          }
196          }
197          }
```

```

Generator.java_1      Tue Nov 12 07:45:33 1996      1
1  /*
2   * $Header: /ccm/meitca/hsl/zonesagents/skeleton/Generator.java 5
3   * Copyright 1996 Horizon Systems Laboratory.
4   * Mitsubishi Electric Information Technology Center America.
5   * All rights reserved.
6   *
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC USA.
8   *
9   * DESCRIPTION
10  * Agent Skeleton Generator:
11  *
12  * $Log: /com/mitscabs1/zonesagents/skeleton/Generator.java $
13  *
14  * 5    11/11/96 5:16P BillP
15  *     Made correction to company name
16  *
17  * 4    9/30/96 7:20P Walsh
18  *     add javadoc comments
19  *
20  * 3    9/29/96 4:32P Walsh
21  *     Allow agents to be in packages
22  *
23  * 2    9/12/96 11:00a Walsh
24  *     Automatically compile the skeleton file
25  *
26  * 1    9/06/96 4:56P Walsh
27  *
28  */
29 package com.meitca.hsl.zonesagents.skeleton;
30
31 import sun.tools.java.*;
32 import sun.tools.java.*;
33 import java.io.*;
34 import java.util.*;
35
36
37 /**
38  * The Generator class generates an RMI-based invocation skeleton for an
39  * Agent. A Method invocation Skeleton allows the ContainerServer to
40  * dynamically invoke any method of an Agent. The Skeleton serves a
41  * similar purpose as does an RMI skeleton or an RPC server side stub.
42  * Before any Agent can travel, it is requires that a Skeleton be
43  * generated by it. A programmer can generate a skeleton by executing
44  * Generator with a command line like the following: <br>
45  * <pre>
46  * java com.meitca.hsl.zonesagents.skeleton.Generator agentclassname
47  * </pre>

```

```

Generator.java_1      Tue Nov 12 07:45:33 1996      2

48   * The agentclassname parameter should be the fully qualified class
49   * name of the agent.
50   * @see ConduitServer
51   * @see Agent
52   * @see AgentSkeleton
53   * @author Thomas Walsin
54   */
55   public class Generator {
56       /**
57       * the main method allows Generator to be called from the command
58       * line.
59       */
60       public static void main(String[] args) {
61           // Sanity check the command line arguments...
62           if (args.length < 1) {
63               System.out.println(StringResources.NO_AGENT);
64               System.out.println(StringResources.USAGE);
65               return;
66           }
67           if (args.length > 1) {
68               System.out.println(StringResources.TOO_MANY);
69               System.out.println(StringResources.USAGE);
70               return;
71           }
72       }
73       String agentClassName = args[0];
74       String packageName;
75       String shortClassName;
76
77       int lastSep = agentClassName.lastIndexOf('.');
78       if (lastSep == -1) {
79           packageName = agentClassName.substring(0, lastSep);
80           shortClassName = agentClassName.substring(lastSep + 1, agentClassName.length());
81       } else {
82           packageName = null;
83           shortClassName = agentClassName;
84       }
85
86       try {
87           String skelFileName = shortClassName + ".Skel.java";
88           printStream out = new PrintStream(new FileOutputStream(skelFileName));
89
90
91           String[] methods = findAgentMethods(agentClassName);
92           printStream out = new PrintStream(new FileOutputStream(skelFileName));
93
94

```



Generator.java 1 Tue Nov 12 07:45:33 1996

- 151 -

```
Generator.java_1      Tue Nov 12 07:45:33 1996      5

188     if (argTypes.length == 0) {
189         if (fieldDef.getType() .getReturnType() == Type.tvoid) {
190             agentMethods.addElement(fieldDef.getName() .toString());
191         }
192     }
193     fieldDef = fieldDef.getNextField();
194 }
195
196     //turn it into an array
197     String[] array = new String[agentMethods.size()];
198     for (int i=0; i<agentMethods.size(); i++) {
199         array[i] = (String)agentMethods.elementAt(i);
200     }
201
202     return array;
203 }
204
205 static void compileSkel(String skelfileName) {
206     Main compiler = new Main(System.out, "javac");
207     String[] params = new String[1];
208     params[0] = skelfileName;
209     if (!compiler.compile(params)) {
210         System.err.println(StringResources.COMPILE_ERROR + skelfilename);
211     }
212 }
213
214 }
```

- 152 -

## StringResources.java\_1      Tue Nov 12 07:45:33 1996

```

1  /*
2   *      $Header: /ccm/meitca/hsl/zones/agents/skeleton/StringResources.java 3      11/11/96 5:16p Billp $
3   *
4   *      Copyright 1996 Horizon Systems Laboratory
5   *      Mitsubishi Electric Information Technology Center America
6   *      All rights reserved.
7   *
8   *      CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA
9   *
10  *      DESCRIPTION
11  *      The String Resources for the skeleton package
12  *      $Log: /com/meitca/hsl/zones/agents/skeleton/StringResources.java 5
13  *
14  *      3 11/11/96 5:16p Billp
15  *      * Made correction to company name
16  *
17  *      2 9/12/96 11:00a Walsh
18  *      * Automatically compile the skeleton file
19  *
20  *      1 9/05/96 4:54p Walsh
21  *
22  */
23 package com.meitca.hsl.zones.agents.skeleton;
24
25 class StringResources {
26
27     // Generator.java
28     public static String NO_AGENT = "usage: generator agentclassname";
29     public static String NO_ARGUMENT = "generator: You must specify an agent class name ";
30     public static String TOO_MANY = "generator: Too many parameters specified.";
31     public static String NOT_FOUND = "generator: Agent class not found --";
32     public static String IO_ERROR = "generator: IO error while attempting to write the skeleton file --";
33     public static String COMPILE_ERROR = "generator: Anb error while attempting to compile the skeleton file --"
34 }

```

- 155 -

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

collaborate with a specific Agent regarding a certain subject is not known a priori, but is, rather, determined as a result of the Agent's computations.)

Both the interaction between these Agents and the context in which they execute must be formally specified. Standardization has not yet been achieved for these objectives. However, the prevalent means for accomplishing these results is via implementing a user model and specifying queries and assertions by using a knowledge representation language, the most popular of which is currently Knowledge Query Manipulation Language (KQML), and by developing Ontologies, or vocabularies of common terms to be utilized by interacting Agents. KQML is comprised of both a language, with a syntax unlike procedural or object-oriented languages, and a communications protocol. An Ontology is defined as "an explicit specification of a conceptualization". The syntax and semantics of an Ontology are generally extensions of a first-order predicate calculus, typically the Knowledge Interface Format (KIF).

The combination of KQML, KIF, and Ontology development is a complex task. Some implementations utilizing these mechanisms are being developed in the Artificial Intelligence community. But this approach has not caught on in other software applications.

In contrast to Intelligent Agents, Mobile Agents are usually explicitly programmed to accomplish a specific task. As an example, a Mobile Agent could migrate to travel agencies on the World Wide Web and determine the lowest cost air fare between two cities. In this case, there is no need for learned behavior or complex reasoning.

However, the amount of information accessible on distributed systems, including databases, the Internet and Intranets, is growing at a staggering pace and much of it is available in incompatible storage formats. The need to filter information and make decisions based on its content is increasing, and with it, the need for increased collaboration among Mobile Agents.

The need for collaboration is illustrated by extending the above travel agency example. A Mobile Agent may need to clone itself and instruct each of its counterparts to migrate to a different travel agency site to obtain the best package for a trip to one or more ski resorts meeting certain parameters, including air fare, lodging, meals, lift tickets, and car rental. Furthermore, the information may, itself, be stored in different formats (databases, etc.), necessitating some amount of filtering and data reformatting. Once the Agents have obtained their information, they need to collaborate with each other to correlate their results and determine the best package available at each resort for different travel dates and the overall best deal.

Mobile Agents are evolving to include increased intelligence and collaboration. Some research organizations have begun incorporating KQML and Ontologies into Mobile

Mitsubishi Electric ITA Confidential

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

Agent implementations, while others utilize different knowledge representation languages to implement Agent communication. Still, others use proprietary object languages.

### **3. Problems with Prior Art**

The KQML/Ontology mechanism and other knowledge representation languages are ill-suited for general-purpose software applications. They require programmers to learn multiple languages to formally specify communications and to develop complex dictionaries of terms - a task that is currently ill-defined. The knowledge representation languages are also extremely unlike the procedural and object-oriented languages utilized in building business and general-purpose software applications.

The need to support asynchronous collaboration adds significant complexity to Agent systems implemented via knowledge representation languages (e.g. KQML) and Ontologies. Furthermore, there exists a class of Mobile Agents which solely require synchronous collaboration, e.g., the travel agency application described above. This type of Agent application essentially subdivides a larger task into smaller pieces and creates Agents to migrate throughout the network to accomplish it. For these Agents, there is no benefit in the added complexity of coding in multiple languages and developing Ontologies, nor in the support for asynchronous communication. A synchronous collaboration framework in the Agent's language is the best means for these Agents to coordinate their results and implement adaptive behavior.

### **4. Some Details of the Invention**

The dynamic synchronous collaboration framework utilizes a distributed synchronization point plus object-oriented abstractions to provide synchronous collaboration for Mobile Agents in a distributed system. The framework simplifies collaboration for applications which subdivide a complex problem into several pieces and farm out the work to multiple Mobile Agents.

This framework enables Mobile Agents within an application to perform synchronous collaboration with one or more affiliated Agents in the application's native language. This is accomplished via the *Agent Group* abstraction. Agents wishing to collaborate with each other must belong to one (or more) of the same Agent Groups. The fundamental principle behind Agent Group collaboration is that Mobile Agents performing sub-computations, possibly on different nodes in a distributed system, will need to periodically correlate their results, and potentially adapt their behavior based on the results of the correlation. The Agent Group acts as a distributed synchronization point for this type of synchronous collaboration.

- 154 -

Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

# Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

## 1. Introduction

This document serves as a patent disclosure on the invention entitled *Dynamic Synchronous Collaboration Framework for Mobile Agents*. This document includes a discussion of: prior art, problems with prior art, details of the invention, and improvements of this invention over prior art.

## 2. Prior Art

“Agent” is one of the most overused terms in the software industry today. Although the term may be used in a wide variety of contexts, software Agents can be classified into two major categories: Intelligent Agents and Mobile Agents. Although the two differ in scope and capabilities, both types of Agents are able to perform work asynchronously and autonomously on behalf of their owners. Intelligent Agents are endowed with some degree of reasoning and learned behavior and may make decisions on behalf of their owners. These Agents are generally stationary and execute on a single system. Mobile Agents, on the other hand, may be transported across nodes in a network, performing computations as they migrate.

The original concept for Agents stems from the Artificial Intelligence community. The behavior of Intelligent Agents may be determined by programming or by decisions made by the Agents on behalf of their owners. As an example, an Intelligent Agent may search a database for unusual outdoor scenery, making its own decisions as to whether or not each scene meets the specified criteria.

Intelligent Agents generally employ both synchronous and asynchronous modes of communication. Agent communication may take the form of queries or collaborative computations. In the realm of Agent computing collaboration may be defined as “the coordination of the computations of multiple Agents to achieve a desired result” (e.g., to solve a complex problem). Collaboration often results from discovery (i.e. the need to

- 153 -

# Mitsubishi Electric ITA

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

*November 12, 1996*

*Horizon Systems Laboratory  
Mitsubishi Electric ITA  
1432 Main Street  
Waltham, MA 02154*

Mitsubishi Electric ITA Confidential

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

The dynamic nature of the Agent Group enables Agents to join and leave the group at will. Hence no a priori knowledge of the actual Agents comprising a group is required and group membership may change over time.

A distributed name space maintains a reference to each Agent Group. Hence, Agent Groups may be located via their entry in the name space. When an Agent Group is created, it is automatically added to the name space.

The Agent Group's primary responsibility is to facilitate synchronous collaboration for Mobile Agents. However, it also serves as a central point for distributing asynchronous events to Agents in the group. A typical event may be the termination of a particular Agent. As a side effect, the Agent Group also tracks the travels of its constituent Agents throughout the network.

The travel agency example illustrates the function of Agent Groups. The problem to be solved can be stated as follows: A user wishes to determine the best package for a trip to a ski resort. The Agents must determine the best travel date and time (e.g., any week in February starting on a Saturday, and flying before noon) and lodging must meet certain criteria (e.g., a two-bedroom condominium). The trip includes all expenses (e.g., air fare, lodging, meals, transportation, and lift tickets). The user may also specify several possible travel destinations.

This application is composed of multiple Agents -- each is responsible for querying one or more data sources (in perhaps different formats) for specific information (e.g., searching a particular travel agency's database of promotional trips to determine if any meet the criteria specified by the user.) Some Agents may query legacy database systems, whereas others may search ODBC compliant databases. (The actual mechanisms for handling different data sources is beyond the scope of this invention.)

Before the Agents are transported throughout the network they join a new Agent Group. The Agent Group tracks its Agents' travels throughout the network. At any point, an Agent may be in one of several states. Figure 1 depicts three Agents in an Agent Group performing computations and one Agent in-transit between nodes.

At different points during their migration, Agents may wish to correlate their results, and potentially change their migration plans or behavior. Suppose each Agent migrates to a local travel agency. Before migrating to travel agencies in different cities, the Agents may wish to correlate their results and potentially adjust their behavior. Suppose one of the Agents determines that a particular ski resort, say Taos, has no two-bedroom condominiums available during February. The Agents in the group may wish to drop all further queries about trips to Taos.

Agent Group collaboration is implemented via a distributed synchronization point (also known as a collaboration point) and a software routine to analyze the results of each

- 8 -

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

Agent's computation. The Agent Group abstraction provides the distributed synchronization point. Each application need only provide its own software method to analyze the results and potentially allow each Agent to adapt its behavior.

The distributed synchronization scheme requires that each Agent arrive at the collaboration point before collaboration may commence. Hence, it is best suited to applications that subdivide a complex problem into many sub-tasks that need to correlate their results. As each Agent arrives at the collaboration point, it posts the results of its computations to the Agent Group, and blocks until all the other Agents in the group arrive. Figure 2 depicts one Agent in the group waiting to collaborate while three others are still performing computations.

The Agent Group collects the results of each Agent's sub-computation. When all Agents have arrived at the collaboration point, as shown in Figure 3, the Agent Group notifies them that collaboration may commence. When it unblocks the waiting Agents, the Agent Group passes the collected results to each Agent. Each Agent then calls the application-specific method to analyze the results and potentially adapt its behavior. Figure 4 shows that all Agents have arrived at the collaboration point and are now in the analysis stage.

The synchronous collaboration framework supports both parallel and serialized collaboration analysis. Figure 4 depicts a parallel analysis implementation; Figure 5 illustrates a serialized collaboration analysis, in which one Agent is in the Analysis state, while the rest remain waiting in the Collaborate state.

The distributed collaboration point may be implemented via an object transport or by an RPC mechanism. It may also be implemented in any procedural or object-oriented language. Object-oriented languages simplify the implementation, as the application need only subclass the Agent Group abstraction and provide a concrete method to analyze the results and potentially adapt the Agent's future behavior.

The Agent Group ensures that all Agents in the group arrive at the synchronization point by tracking the Agent's migration through the network. In the current implementation, as part of a mobile Agent system, a server (known as the Conduit Server) on each node manages the Agent's migration and informs each of its Agent Groups when the Agent arrives and departs. Should an Agent fail to arrive at a collaboration point in a pre-determined, configurable length of time, the Agent Group will "ping" the Agent by sending a message to ensure that it is still alive, as shown in Figure 6. (In the current implementation, this is performed via the Conduit Server on the node where the Agent is executing.) If the Agent fails to respond to the ping, the Agent Group notifies the other members of the group by distributing a Collaboration Failed asynchronous event to them that indicates that a particular Agent is not responding. The Agents may then indicate whether or not they wish to proceed with the collaboration. Figure 7 depicts the Agent Group's distribution of a Collaboration Failed event followed by the remaining Agents

- 155 -

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

indicating that they wish to continue the collaboration. Alternatively, the Agents could decide to abort the collaboration.

The Agent Group also implements deadlock detection by means of the time-out mechanism and Agent ping described above. The ping actually returns the state of the Agent in question (via the Conduit Server). If the Agent is already in the Collaborate state but has not arrived at this collaboration point, a deadlock has occurred. This is shown in Figure 8. In this case, the Agent Group will abort the collaboration and notify the Agents in the group of the deadlock by sending them a Deadlock Event, as shown in Figure 9. Note that since Agent Group collaboration is designed for closely coordinated Agents, the existence of a deadlock is generally a result of a programming error. Hence, the Agent Group does not need to employ a more sophisticated scheme for deadlock detection or avoidance.

Agent Groups provide reliability by saving the state of each Agent in the group to persistent storage. If the Agent Group should terminate unexpectedly, it is restarted and reads the state of the group members from persistent storage. The Agent Group's restart is transparent to the Agents. They actually communicate with an Agent Group Proxy (instead of an Agent Group), which shields them from Agent Group failures.

Figure 11 illustrates the flow of a typical collaborating Mobile Agent. Each Agent creates an Agent Group Proxy for each Agent group it wishes to join. The proxy's primary responsibility is to maintain an active connection to its associated Agent Group and to re-establish the connection or re-create the Agent Group, as needed, during Agent Group failures.

When the Agent Group Proxy is initialized, it establishes a connection with the appropriate Agent Group. It first attempts to find a reference to it in the name space. If this fails, it creates the Agent Group and enters a reference to it into the name space. The algorithm guarantees the atomicity of this operation.

After the proxy has created the Agent Group or established a connection to it, the Agent continues its computations and migrations. Eventually, it may attempt to collaborate its results. This is accomplished by means of the Agent Group Proxy. If the Agent Group has failed or is in an unknown state, the proxy will be unable to contact it. If this occurs, the Agent Group Proxy retries the communication. If, at this point, the proxy is still unable to communicate with the Agent Group, it recreates it and updates the name space with a reference to the new Agent Group. The re-creation of the Agent Group is serialized to prevent multiple Agent Group Proxies from simultaneously doing so. If a proxy determines that the Agent Group is in the process of being re-created, it simply waits until it can find a reference to the new group in the name space. Once the Agent Group has been re-created, the proxy retries the collaboration.

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

As mentioned earlier, Agent Groups provide a distributed events mechanism in addition to the synchronous collaboration framework. The distributed events mechanism enables the Agent Group to notify members of the group of outstanding events, such as the termination of an Agent, or the detection of a collaboration deadlock. Agents may build on this mechanism to forward application-generated events to other members in the group. Hence, if an Agent catches an application-generated exception it may notify the other Agents in the group before terminating. Figure 10 illustrates how the Agent Group forwards an Exception Event, generated by one of its Agents, to the rest of the group.

Currently, the Agent Group handles distributed events by maintaining a (possibly remote) reference to an event queue associated with each Agent. The Agent Group forwards all events it receives to the other members of the group by enqueueing it. The Agent Group may be implemented without a remote referencing capability, but this would add significant complexity. In the current implementation, each Agent contains an events thread which is responsible for dequeuing the event and calling the application-specific event handler. (This thread could easily be a process, if threads were not available.)

In addition to forwarding events to members of the group, the Agent Group may generate its own events and distribute those to the group members. As an example, the Agent Group distributes a *Deadlock* Event to the group members when it detects a deadlock situation, as illustrated in Figure 9.

As mentioned earlier, Agent Groups track the migrations of their constituent Agents, with help from the Conduit Server (in the current implementation). If an Agent is in transit when the Agent Group is forwarding events, it queues the events locally, as shown in Figure 10. Once the Agent arrives at its destination, it notifies the Agent Group, via the Conduit Server. The Agent Group then flushes the events it stored locally for that Agent.

Agent Groups implement varying levels of persistence, as defined by the actual configuration. Memberships and Agent states are updated infrequently, so they are generally written to persistent storage. However, local queues maintained on behalf of in-transit Agents may also be saved in persistent storage, if desired.

Implementation of the Agent Group in an object oriented language enables applications to extend its functionality, if required, by subclassing the Agent Group object.

## 5. Benefits of the Invention

The work described above provides several benefits. It greatly simplifies the implementation of synchronous collaborating Mobile Agents, especially for applications that subdivide a complex task into a series of related sub-tasks. There is no need for application developers to learn knowledge query representation languages, such as KQML, or to develop complex Ontologies. The dynamic synchronous collaboration framework provides a natural programmatic implementation that may be programmed in a

- 161 -

**Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents**

number of languages. An added benefit of this scheme is that collaboration is implemented as part of the application, programmed in the same language as the rest of the application. The current implementation in an object-oriented language (Java) also enables the technology to be easily extended so that more complex intelligence schemes may be supported.

Agent Groups facilitate Mobile Agent tracking and may also be queried by tools that monitor Agents. In addition, collaboration may be performed through discovery, as with many knowledge representation languages.

Agent Groups also offer reliability by providing persistent Agent Group membership and Agent state information. Furthermore, Agent Group Proxies shield Agents from the effects of Agent Group failures. They detect an Agent Group's failure and automatically restart the failed Agent Group (which then restores the state of its membership from persistent storage) and retry the current operation (e.g., collaboration). The algorithm also guarantees the atomicity of Agent Group creation and re-creation.

An Agent Group also detects Agent failures and deadlocks and forwards this information to the other members of the group via its distributed events mechanism. Furthermore, this mechanism allows Mobile Agents to asynchronously notify associated Agents of relevant events, such as the catching of an exception.

Implementation of Agent Groups in an object-oriented language also enables additional application-specific functionality to be easily added via inheritance and polymorphism.

Mitsubishi Electric ITA Confidential

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

## 6. Figures

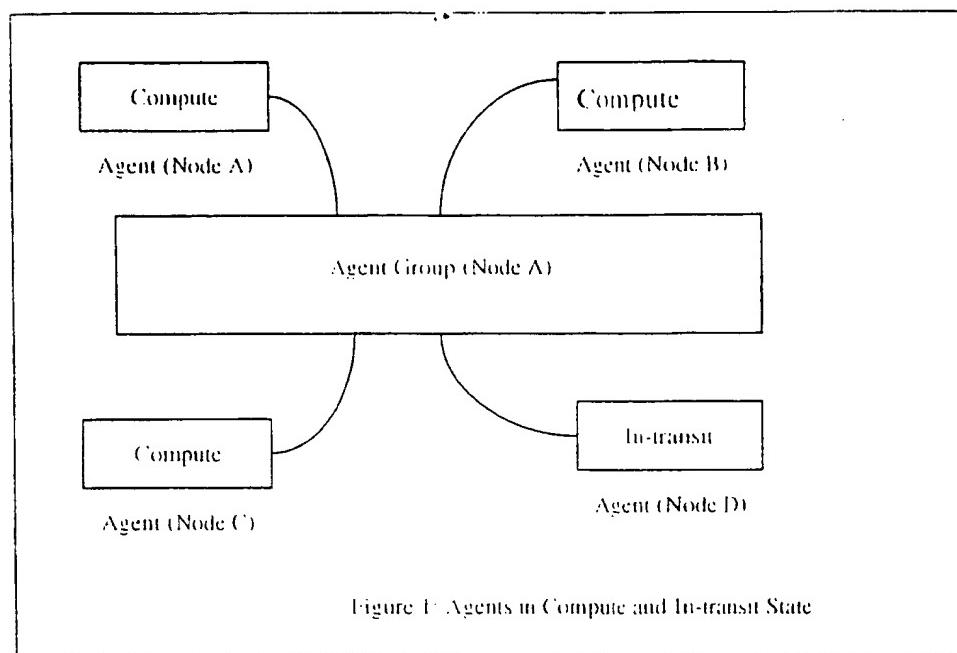


Figure 1: Agents in Compute and In-transit State

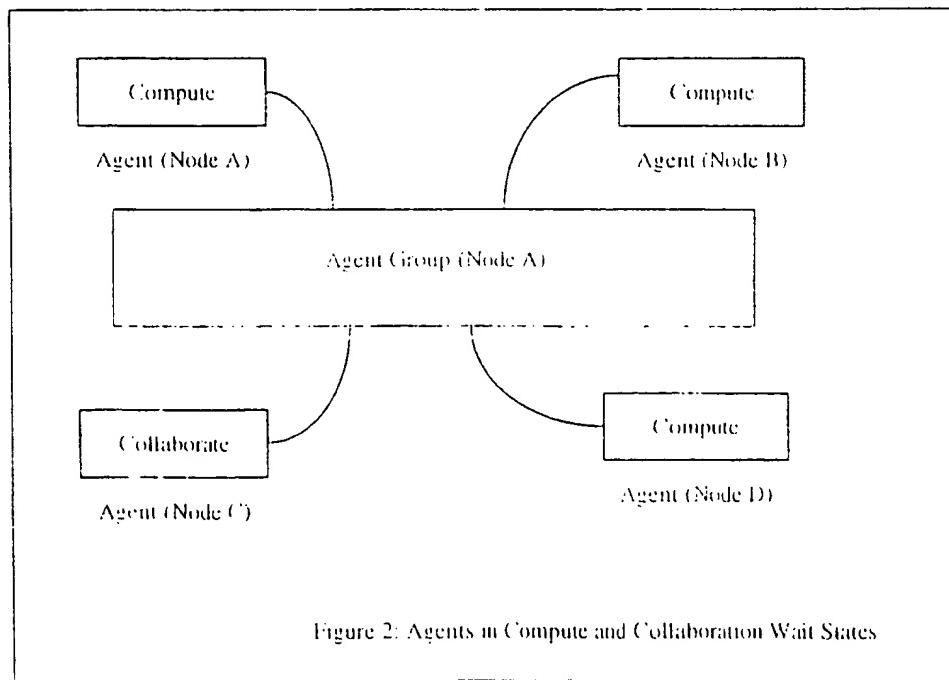
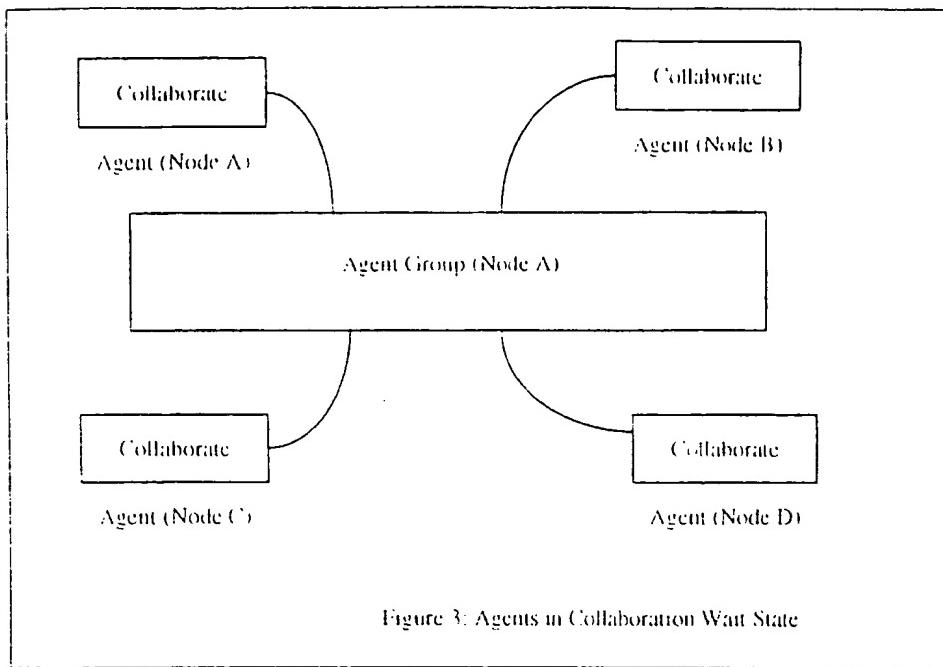


Figure 2: Agents in Compute and Collaboration Wait States

Mitsubishi Electric ITA Confidential

- 163 -

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents



- 164 -

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

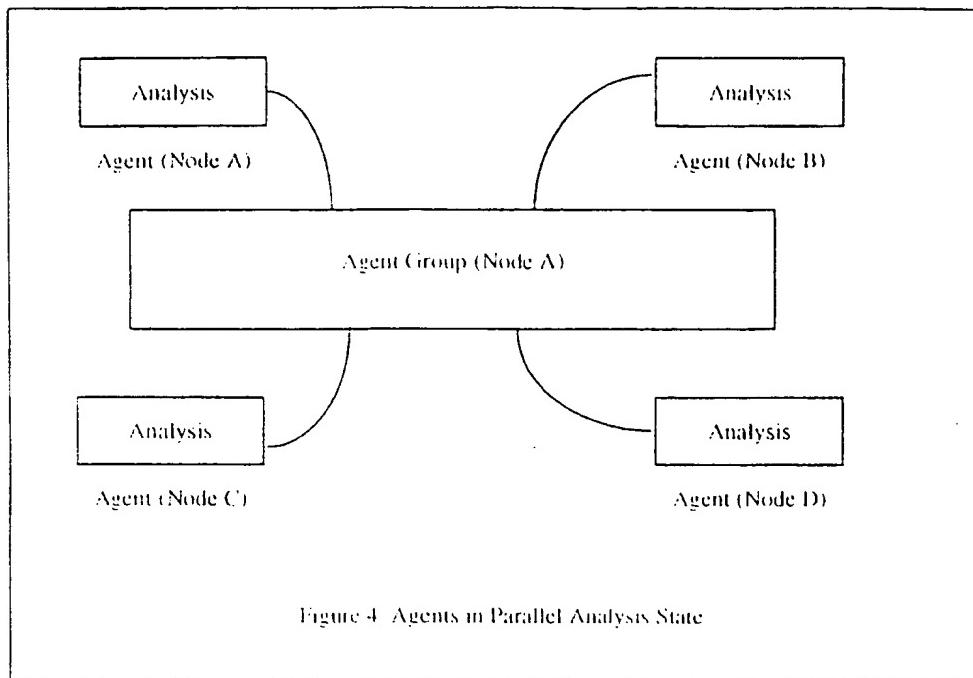


Figure 4: Agents in Parallel Analysis State

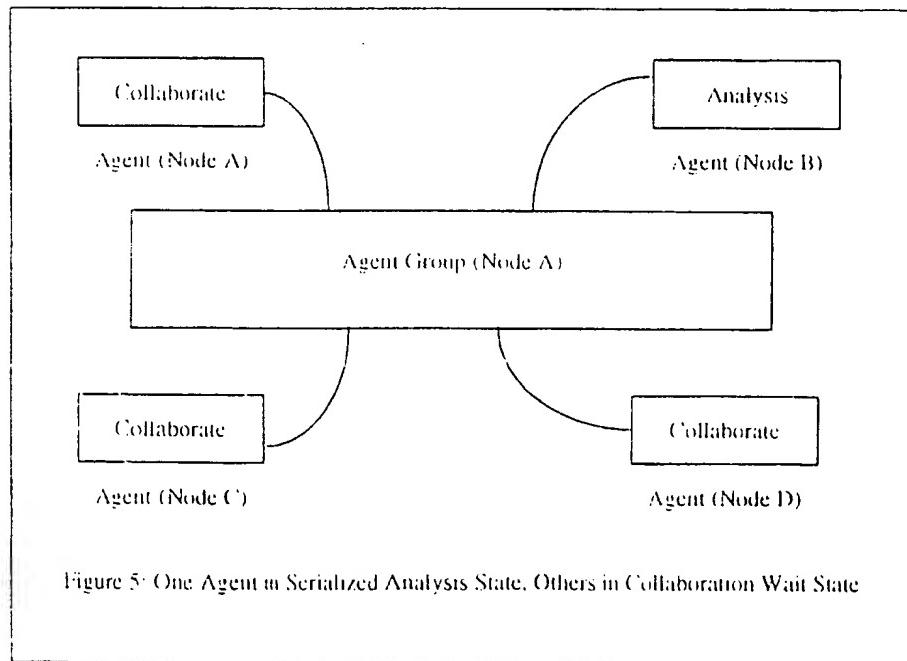


Figure 5: One Agent in Serialized Analysis State, Others in Collaboration Wait State

Mitsubishi Electric ITA Confidential

- 165 -

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

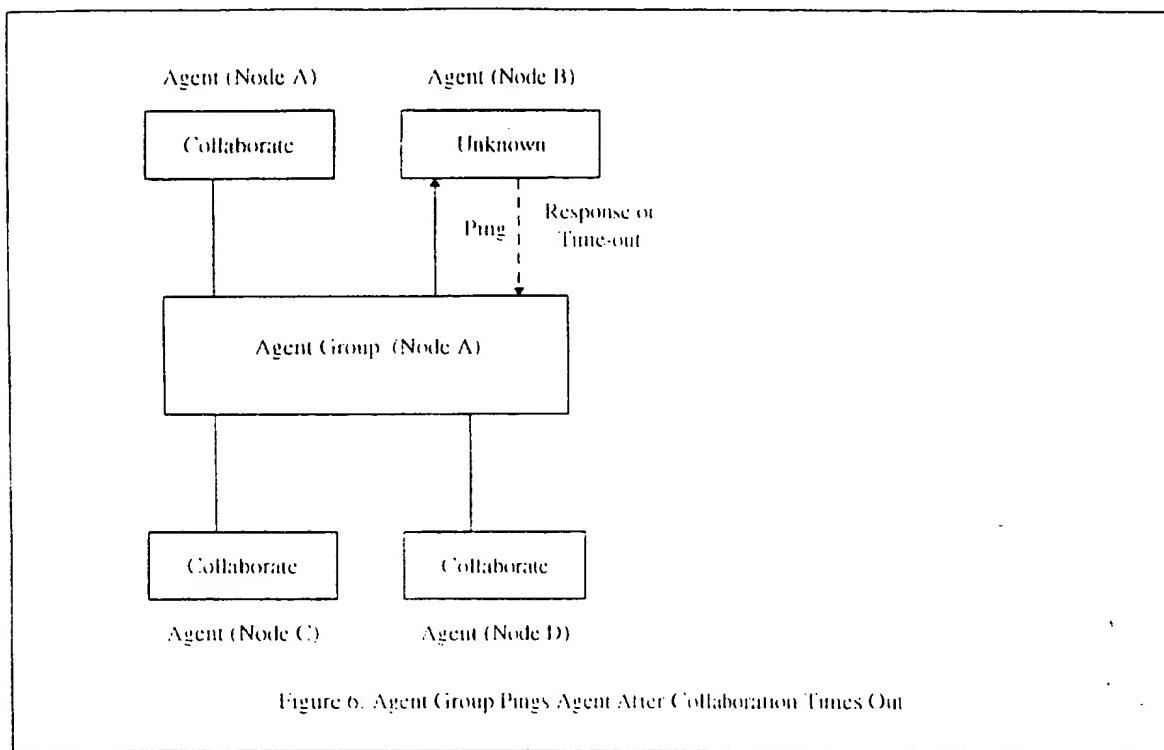


Figure 6. Agent Group Pings Agent After Collaboration Times Out

- 166 -

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

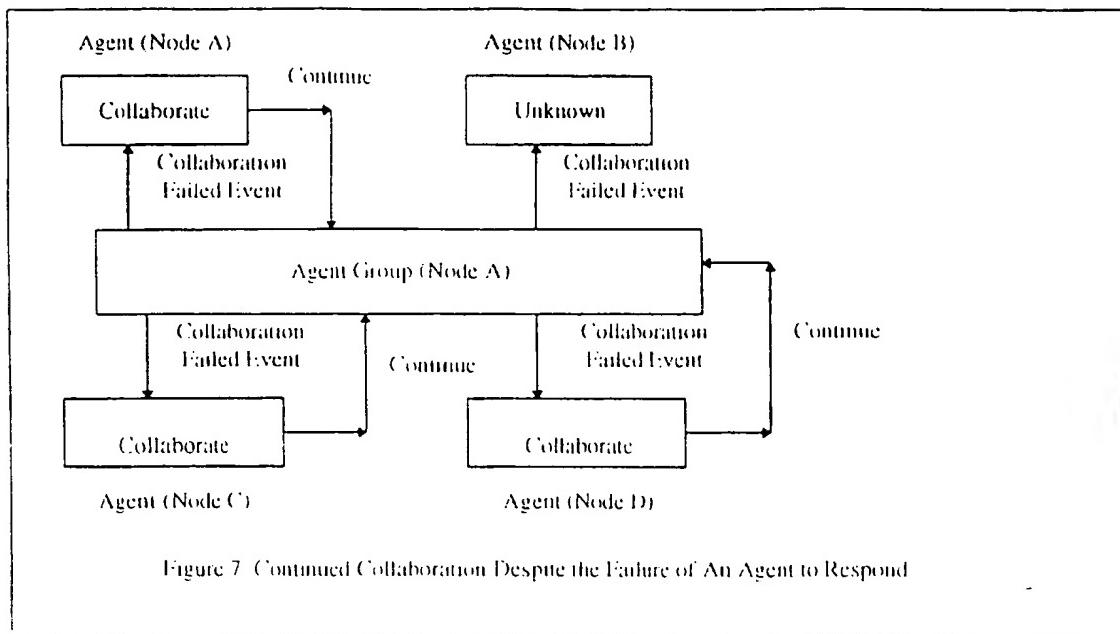


Figure 7 Continued Collaboration Despite the Failure of An Agent to Respond

- 167 -

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

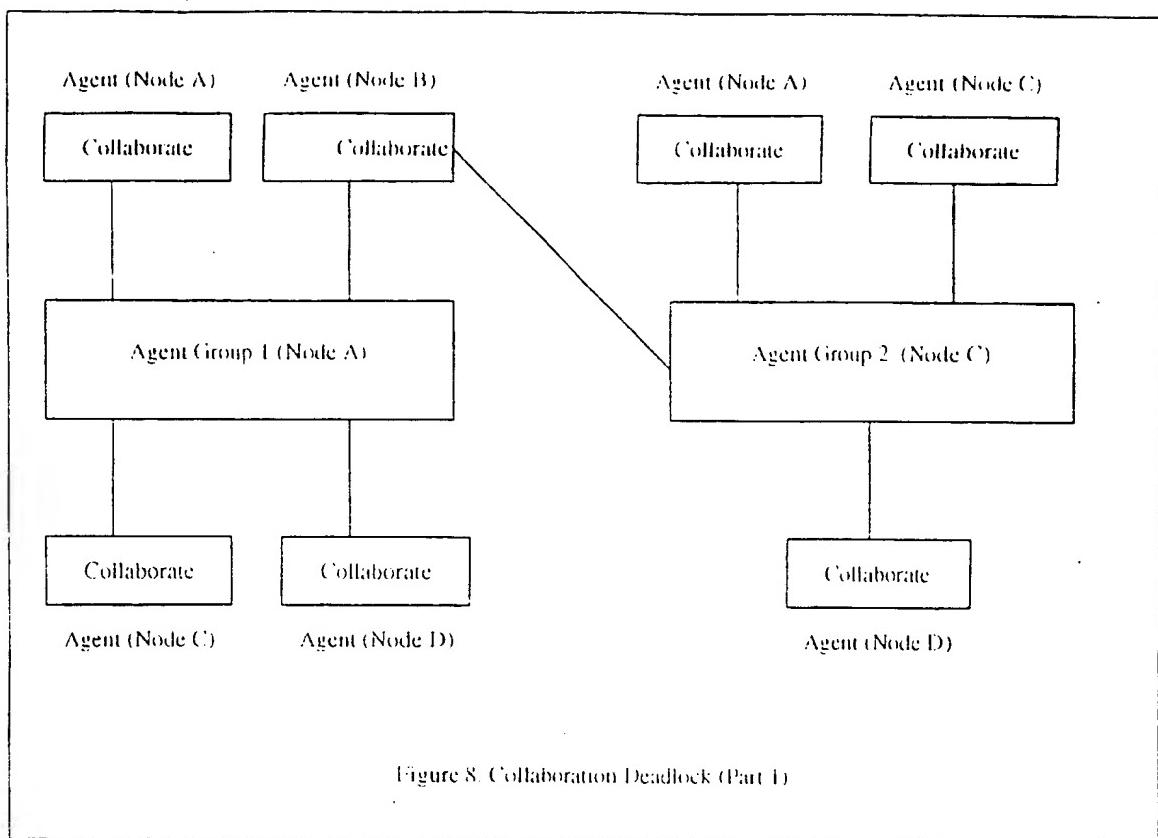
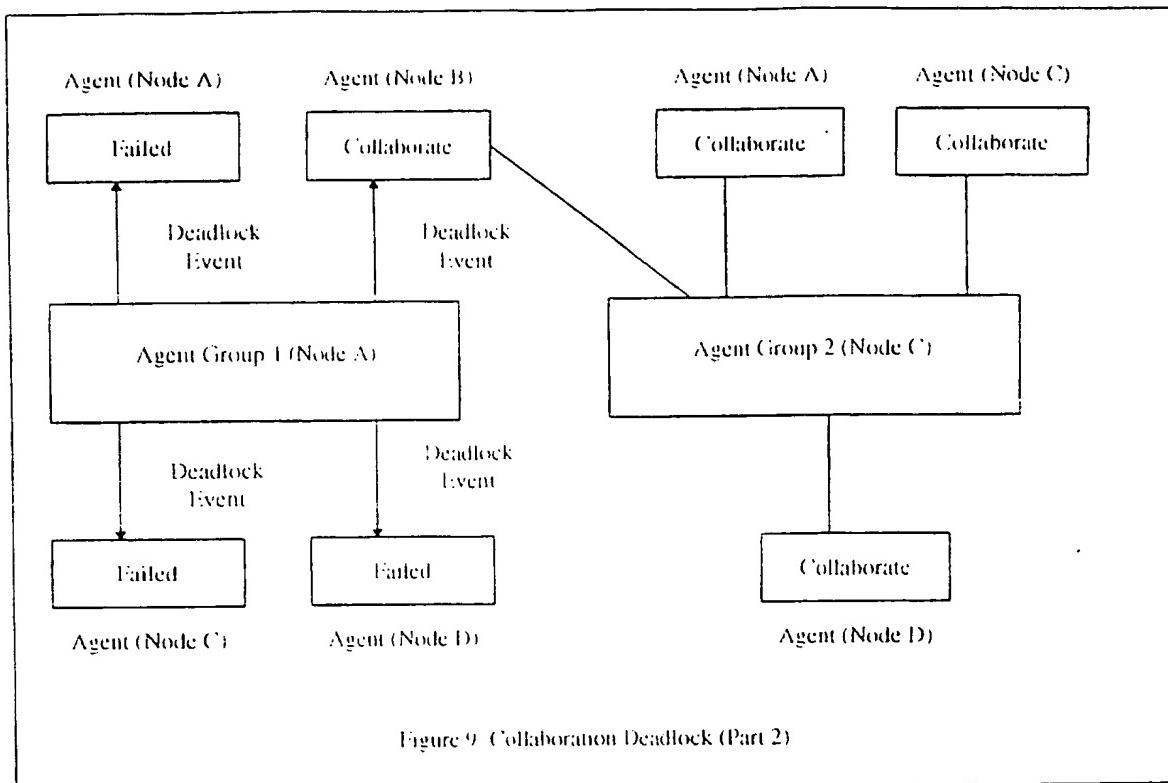


Figure 8. Collaboration Deadlock (Part 1)

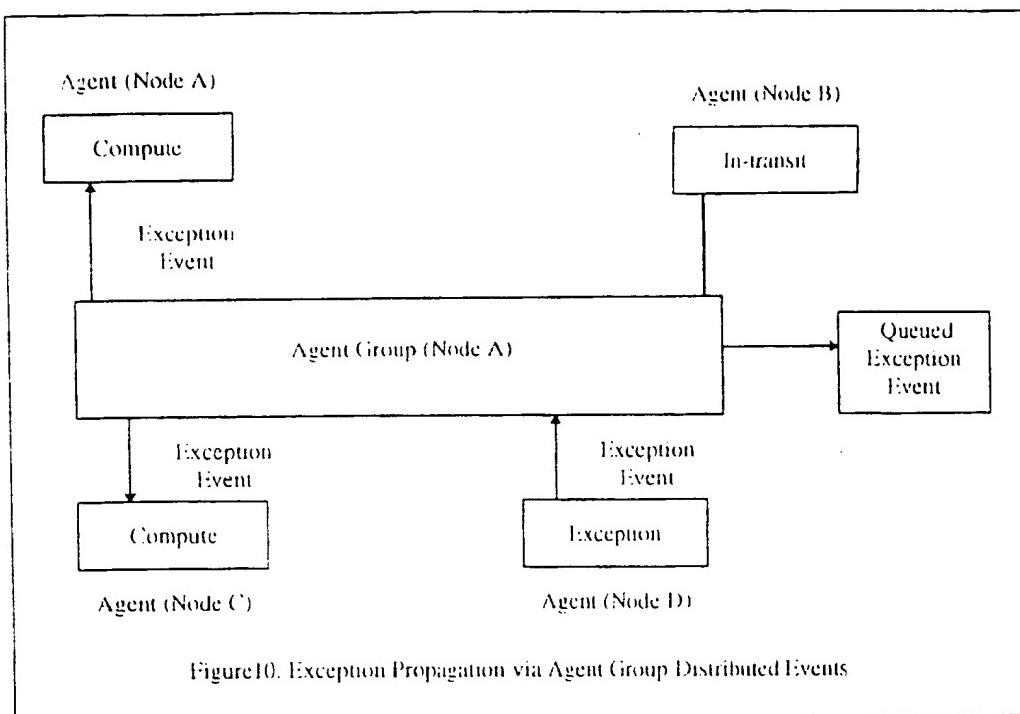
-168-

Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents

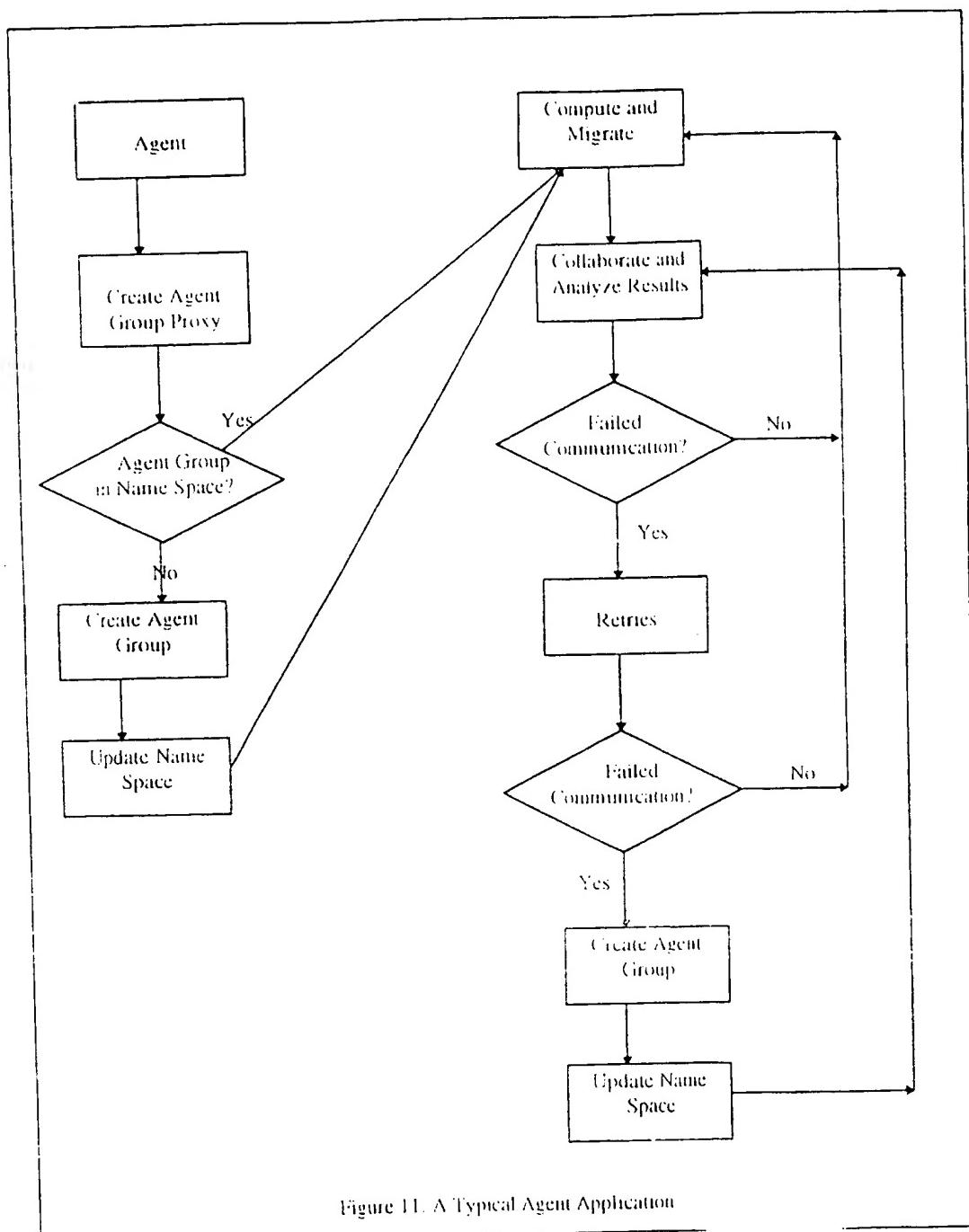


-169-

## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents



## Patent Disclosure for Dynamic Synchronous Collaboration Framework for Mobile Agents



Mitsubishi Electric ITA Confidential

-171-

# Mitsubishi Electric ITA

## Design Specification for Java Agent Coordination

*Version 0.2*  
*November 12, 1996*

*Horizon Systems Laboratory*  
*Mitsubishi Electric ITA*  
*1432 Main Street*  
*Waltham, MA 02154*

Mitsubishi Electric ITA Confidential

# Design Specification for Java Agent Coordination

*Version 0.2<sup>1</sup>*

## 1. Introduction

Java agents have the capability of coordinating their activities. This coordination takes two main forms: synchronized collaboration and asynchronous notifications. These modes of communication complement each other and their implementations are related.

### Collaboration

The collaboration paradigm is implemented as follows:

- multiple agents perform computations at different nodes in the network;
- after performing all or a subset of their operations, agents wait at a distributed synchronization point until all agents have "arrived";
- agents coordinate by potentially serialized execution of a synchronized method.

Coordination can take the form of performing computations on intermediate results and/or updating agent itineraries.

### Notification

The notification scheme uses a distributed events mechanism to communicate agent state changes that occur asynchronously. It is mainly utilized for distributed exception handling, but may also be used to notify related agents of events, such as another's termination.

As described above, agent coordination utilizes distributed synchronization and event handling. However, Java possesses neither of these. Actually, its synchronization primitives are limited to critical sections on a single system and a notification scheme that can only handle a single type of event. Hence, a great deal of the complexity in the task of agent coordination is comprised of implementing mechanisms to overcome Java's unsuitability for distributed processing. These mechanisms are described in the remainder of this document, along with the new objects that implement them.

---

<sup>1</sup> Revision History:

07/26/96 Version 0.1 Initial revision.

11/12/96 Version 0.2 Major changes to Notifications and Exceptions and Collaboration sections to reflect actual implementation. Removed Events section.

## 2. Synchronization

As mentioned above, agent coordination requires distributed synchronization mechanisms, such as locks and events. Since Java does not provide any distributed locking, we are faced with the following choice:

1. Utilize a centralized locking scheme.
2. Develop our own distributed locking paradigms
3. Use Java's native methods to interface with the operating system's synchronization primitives.

The latter choice is unacceptable because it compromises the portability of our agent infrastructure, whereas the first option may potentially introduce considerable network overhead. The second alternative also requires significant development effort even to implement a well-known solution to the problem.

In the interest of expediency (as required by the tightness of our schedule), the initial Java agent infrastructure utilizes RMI to implement a centralized locking scheme. Locking is implemented via method invocations on a potentially remote object (called an *AgentGroup*. See Section 3 for details.) Agent coordination is implemented by (remote) invocations on a method which requires synchronized access. (See Section 3.)

## 3. CollaboratorAgent and AgentGroup

Agents that may wish to collaborate their results subclass *CollaboratorAgent*, an abstract class derived from *Agent*. *CollaboratorAgent* contains a minimum of two threads – one to perform computation and to travel and another to handle asynchronous event notifications. Applications that contain collaborating agents subclass *CollaboratorAgent* and, of course, these derived classes may implement several computation threads.

Each collaborating agent belongs to one or more agent groups, which are described by *AgentGroup* objects. *AgentGroup* is actually a distributed RMI object. *AgentGroup* defines the interface, *AgentGroupImpl*, an abstract base class derived from *PersistentEventGroupImpl*, provides the implementation. (See *Design Specification for Distributed Java Events* for a description of *PersistentEventGroupImpl*.)

An *AgentGroup* is essentially a unit of coordination. Hence agents may coordinate with all the other agents in an *AgentGroup*. They may also create new *AgentGroups* on demand, and join and leave groups at will. (In practice, *AgentGroup* must be subclassed to implement agent collaboration. However, this document does not distinguish between the base class and its derived classes and refers to them collectively as *AgentGroups*.)

An *AgentGroup* is an object that is essentially shared among the members of an agent coordination group. It is owned by the *Agent* that initially creates the group (an *agent group* is created by instantiating *AgentGroup*) and accessed via a potentially remote RMI reference.

The relationship between *CollaboratorAgent* and *AgentGroup* is similar to that of *Thread* and *ThreadGroup*: A *ThreadGroup* maintains references to the *Threads* composing the group; an *AgentGroup* maintains a similar list of its *CollaboratorAgents*. Both *ThreadGroups* and *AgentGroups* form a hierarchy and each contains a reference to its parent group. *Thread*'s constructors are either passed a reference to a *ThreadGroup* or they save a

- 174 -

## Mitsubishi Electric ITA Confidential

reference to the parent's Thread's group. Likewise, CollaboratorAgent's constructors may be passed one or more AgentGroups or save a reference to AgentGroup of its parent, if it exists.

However, there is one noticeable difference between the two types of groups. A Thread may only belong to one ThreadGroup at a time, whereas a CollaboratorAgent may simultaneously be a member of several AgentGroups, which constitute a hierarchy. Hence, Threads contain a reference to their associated ThreadGroup, but CollaboratorAgents maintain a list of references to their AgentGroups.

Support for multiple AgentGroups facilitates coordination among related groups of agents and also enables the creation of an AgentGroup hierarchy through which communication may be propagated. The pilot application may utilize multiple agent groups, as shown in Figure 1.

This figure portrays an example of two AgentGroups. AgentGroup A consists of three agents – one on each LAN in the system, whereas AgentGroup B contains agents on multiple nodes on LAN 3. A distributed application running on a configuration such as this typically creates multiple AgentGroups when it needs to traverse the WAN. In this case, 3 agents are created in AgentGroup A, one for each LAN. Once they arrive on a node, these agents clone themselves, creating more agents to be distributed across nodes on the LAN (e.g., AgentGroup B). This approach minimizes network traffic over the WAN and forms a multi-level hierarchy of AgentGroups. Agents in the intra-LAN groups (e.g., AgentGroup B) can communicate their results, which can then be coordinated with the inter-LAN group (i.e., AgentGroup A).

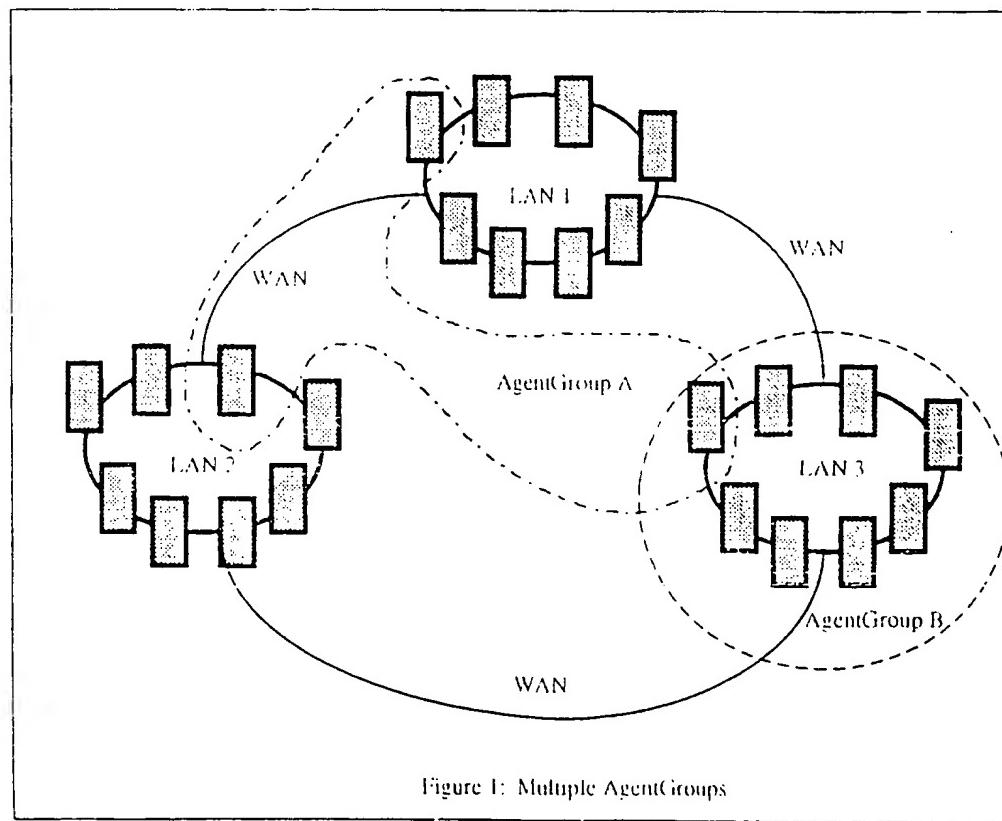


Figure 1: Multiple AgentGroups

- A7S -

**Mitsubishi Electric ITA Confidential**

By default, RMI passes all local objects by copy. However, the above scheme requires that each Agent actually maintain (remote) references to its AgentGroups. Hence, the agent infrastructure overrides some of RMI's object serialization methods to generate the required references.

#### 4. Notifications and Exceptions

Both asynchronous notification and collaboration are implemented via the associated AgentGroup object. The following description serves as an example of asynchronous notification handling via distributed events. (See *Design Specification for Distributed Java Events* for more details on event handling.)

- 1 An agent catches an exception and wishes to notify the other members of the group
- 2 The agent posts an *AgentExceptionEvent* to the AgentGroup via the group's *postEvent* method. The *AgentExceptionEvent* indicates the type of exception that occurred
- 3 The AgentGroup posts the *AgentExceptionEvent* to the other agents in the group
- 4 Each agent's Event Handler performs application-specific handling of the *ExceptionEvent*.

If the agent that caught the exception wishes to notify another associated Agent Group, it repeats the above procedure with the next AgentGroup object.

Occasionally, the Agent Group may generate events that it directly forwards to a member of the group (e.g., to "ping" it after it fails to arrive at a collaboration point) or others that it "multicasts" to its members (e.g., an Agent's failure to reply to a ping).

#### 5. Collaboration

Agent collaboration is more complicated than asynchronous notifications. It is implemented as follows:

- 1 An agent performs some computation and arrives at a synchronization point
- 2 The agent performs a (remote) invocation of the AgentGroup's *collaborate* method, passing it an *AgentStatus* object.
- 3 If no collaboration is in-progress (i.e., this is the first agent that has requested collaboration), collaborate calls its synchronized *beginCollaborate* method, which initializes the number of agents to wait for before collaboration may begin (i.e.,  $n-1$  for an  $n$ -agent group). (Race conditions are handled by checking if the agent count has been initialized (i.e., it is non-zero) before continuing. A race implies that two agents are attempting to initiate the same collaboration. Otherwise the application is semantically incorrect.) This routine saves the requesting agent's unique ID (obtained from the *AgentStatus* object) and starts a collaboration timer (with sufficient time for agents to arrive across the WAN) by passing a configurable time-out value to the AgentGroup's *wait* method.
- 4 Otherwise (i.e., if a collaboration is in-progress), collaborate calls the synchronized *decrementCollaborate* method to add the requesting agent's unique ID to the list of

-176-

**Mitsubishi Electric ITA Confidential**

agents arriving at the collaboration point and to decrement the waiting agent counter. If the counter is non-zero, this method blocks with no time-out value. Otherwise, the Agent Group awakens the other threads waiting for collaboration to commence.

5. If the timer expires, all agents did not arrive at the collaboration point in time. This may occur because an agent received an uncaught exception and terminated unexpectedly or because a deadlock condition exists. (Since agents may belong to multiple groups, overlapping collaborations could result in deadlock. However, a properly coded application will not generate deadlocks because the collaboration points must be known a priori and must be executed in the same order among collaborating agents, i.e., collaboration must first handled at the leaf Agent Group and propagated up the hierarchy.) If a time-out occurs, the Agent Group thread initiating the collaboration (on behalf of its associated CollaboratorAgent) is awakened and performs the following:
  - It calls the AgentGroup's *checkCollaborate* method which returns the collaboration state (either *CollaborationSucceeded* or *CollaborationFailed*, depending on the value of the waiting agent count)
  - If the collaboration failed, the collaborate method calls *recoverCollaborate*. This method attempts to communicate with the agents that did not arrive at the collaboration point by "pinging" them (It determines these agents by comparing the list of arrived agents with the group membership.) The ping is actually performed via a message to the Conduit Server on the node where each agent was last known to be executing. If the agents are located and any of them is in the *Collaborate* state and but has not arrived at this collaboration point, a collaboration deadlock has occurred and the collaboration state is set to *CollaborationDeadlock*. If any of the agents cannot be located, the collaboration state is set to *CollaborationFailed*. Otherwise, it is set to *CollaborationRetry* and the Agent Group thread retries the collaboration up to a configurable number of times (after which the state will be set to *CollaborationFailed* if the collaboration still times out.)
  - The initiating Agent Group thread wakes up the other blocked threads by calling *notifyAll* on the AgentGroup.
6. Once all the collaborating Agent Group threads have been awakened, (either because the counter has dropped to 0 or a time-out occurred), they call the AgentGroup's *checkCollaborate* method. If this method returns a state other the *CollaborationSucceeded*, each thread returns the appropriate exception (i.e., either *CollaborationFailedException* or *CollaborationDeadlockException*) to its invoking CollaboratorAgent.
7. If the collaboration state is *CollaborationSucceeded*, collaboration commences by executing a potentially synchronized application-specific method, *analyzeResults*, to perform the collaboration. *analyzeResults* is an abstract AgentGroup method that must be implemented by derived classes to perform application-specific collaboration. As a result of collaboration, agent itineraries may be updated.
8. As each thread completes its collaboration, it replies to the remote *collaborate* invocation, passing any state specified by the implementation, (e.g., updated itineraries).
9. The agents then continue execution on their current node and travel as required by their itineraries.

- 177 -

Mitsubishi Electric ITA Confidential

As described above, collaboration requires complex mechanisms. However, most of these are necessitated by the limited synchronization primitives provided by Java.

- 178 -

# Mitsubishi Electric ITA

## Design Specification for Distributed Java Events

*Version 0.2*  
*November 12, 1996*

*Horizon Systems Laboratory*  
*Mitsubishi Electric ITA*  
*1432 Main Street*  
*Waltham, MA 02154*

---

Mitsubishi Electric ITA Confidential

- 179 -

Mitsubishi Electric ITA Confidential

# Design Specification for Distributed Java Events

*Version 0.2<sup>1</sup>*

## 1. Introduction

This document describes a general-purpose solution to distributed event management.

## 2. Overview

This section provides a high-level description of the objects that implement the distributed events framework.

### 2.1. Objects

#### Events

Events are described by classes that derive from *EventType*. Each *EventType* object includes a unique ID (a String) generated by the constructors and a description of the event (a String) optionally passed to the constructors.

#### Event Handler

The *EventHandler* interface specifies the *handleEvent()* method which implements an Event Handler. Each application must provide its own Event Handler to perform application-specific event management.

#### Event Notification

The *EventPost* interface specifies the *postEvent()* method which implements event notification. Several of the classes described below implement this interface.

#### Event Queue

Any object receiving notification of events must own an Event Queue and an associated thread that manages the queue. The Event Queue's constructor is passed an EventHandler stub. It instantiates an *EventQueueThread* (a protected class derived from Thread) to manage the queue and passes the EventHandler stub to its constructor. The *EventQueueThread*'s *run()* method removes events from the queue and calls the *handleEvent()* method.

---

<sup>1</sup> Revision History:

09/26/96 Version 0.1 Initial revision.

11/12/96 Version 0.2 Expanded discussion of Event Manager and Event Group. Added Reliability Features and Garbage Collection sections.

## Mitsubishi Electric ITA Confidential

An Event Queue is a distributed RMI object. *EventQueueImpl* implements the *EventPost* interface by appending the specified *EventType* object to the queue and calling the associated *EventQueueThread's notify()* method. This action wakes up the *EventQueueThread* (if it was blocked). The thread then proceeds to dequeue each pending event and pass it to the *handleEvent()* method.

**Event Manager**

The Event Manager handles the registration, posting, and notification of events. When objects request notification of specific event types, they must pass the Event Manager a reference to their *EventPost* interface. The Event Manager saves the reference in a hash table that it uses to map event types to their notification requests. When an object posts an event (via the Event Manager's *postEvent()* method), the Event Manager obtains the list of objects to notify from the hash table and calls each of their *postEvent()* methods.

The Event Manager also maintains a list of objects that have requested notification of all events. Whenever it receives an event, the Event Manager also forwards it to these objects, by calling each of their *postEvent()* methods.

*EventManager* is an RMI distributed object. *EventManager* describes the interface, whereas *EventManagerImpl* provides the implementation. *EventManager* defines methods to manage event registrations (e.g., add and delete registrations).

*EventManagerImpl* also implements the *EventPost* interface. Its *postEvent()* method is passed an event from its originator and notifies the recipients of the event by invoking their *postEvent()* methods.

At most one Event Manager runs on each node in the Agent system. A reference to it can be obtained via the RMI Registry.

**Event Group**

A group of objects interested in a common set of notifications may form an Event Group. *EventGroup* is a distributed RMI object; *EventGroup* defines the interface, *EventGroupImpl* consists of the implementation. *EventGroup* defines methods that manage the group (e.g., add and remove members).

Whereas the Event Manager filters posted events on behalf of its clients, forwarding them only the events they requested, the Event Group performs no filtering. *EventGroupImpl* implements the *EventPost* interface. However, its *postEvent()* method forwards each event it receives to every member of the group (by calling their *postEvent()* methods).

In practice, Event Groups should be used sparingly because they could allow an abusive application to generate unwanted notifications. The Agent framework does utilize this mechanism, but it verifies that all events are generated by a group member and it only utilizes the Event Group for some well-known events, such as the termination of an Agent.

Each Event Group is identified by a unique name either passed to its constructor or assigned by it. Objects may obtain a handle to a particular Event Group by presenting the RMI registry with this unique name.

## 2.2. Reliability Features

The Distributed Events framework incorporates several reliability features. These generally take the form of persistent state and proxy objects.

### Event Manager

The Distributed Events framework shields objects from the effects of Event Manager failures in two ways:

- by saving its state (i.e., event registrations) to persistent storage, and
- by encapsulating its functionality in a proxy object.

The Event Manager saves the following items in the persistent store:

- the hash table mapping specific event types to the objects registered to receive them and
- the list of objects to be notified of all events

Each time an object either registers or "unregisters" its interest in receiving any (or all) events, the Event Manager writes the updated event registrations (i.e., either the hash table or the list of objects, but not both) to persistent storage via the Persistent Storage Manager. These are fairly infrequent operations, so the performance impact of this process should be negligible. (See *Design Specification for Java Agent Persistence Package* for details on the Persistent Store Manager.)

Whenever the Event Manager is started, its initialization code attempts to recover any existing event registrations from persistent storage. Hence, registrations are not lost when the Event Manager fails.

In addition, the Event Manager Proxy also prevents Objects from experiencing the effects of Event Manager failures. It acts as a wrapper object for the Event Manager. Each Object wishing to communicate with the Event Manager must first create an *EventManagerProxy* object. The proxy obtains a reference to the Event Manager via the RMI Registry. If, at any point, the proxy is unable to communicate with the Event Manager, it will attempt to obtain a new reference from the registry. It will retry this operation several times before throwing an exception.

Furthermore, if the Event Manager should fail, it will automatically be re-started by the Java Agent System.

### Persistent Event Group

The distributed Events framework provides a general-purpose implementation of Event Groups (i.e., *EventGroup*), as described above. However, it also provides a reliable implementation of Event Groups: Persistent Event Groups. Persistent Event Group is a distributed RMI object that extends *EventGroup*—the interface is defined by *PersistentEventGroup* and the implementation is provided by *PersistentEventGroupImpl*, a subclass of *EventGroupImpl*. Persistent Event Group utilizes persistent storage in much the same manner as the Event Manager. Whenever an Object joins or leaves the group, it saves a reference to it in persistent storage.

Each Persistent Event Group is also encapsulated in a proxy object: *PersistentEventGroupProxy*. This proxy operates somewhat differently than the Event Manager Proxy, depending on how it is constructed. In some cases, it will attempt to locate a named Persistent Event Group in the RMI registry. If a name is not supplied to the constructor, it will create a new Persistent Event Group and register it.

## Mitsubishi Electric ITA Confidential

Since Event Groups are temporary objects, unlike the Event Manager, which is a persistent server, their proxies cannot expect their associated Persistent Event Groups to be automatically re-started upon failure. Hence, a proxy may re-create the group if it is unable to obtain a reference to it after sufficient retries.

This, however, introduces a race condition wherein multiple proxy objects may simultaneously attempt to re-create the Persistent Event Group. This is handled by the following mechanism:

- If the owner of the Persistent Event Group detects the failure, it re-creates the group. (The owner is defined as the proxy that initially created the group.)
- If the owner does not detect the failure in a configurable length of time, other proxies may attempt to re-create it.
- The Persistent Event Group is re-created via the following steps:
  - Attempt to create a temporary binding in the RMI name space with a well-known name derived from the Persistent Event Group's unique name.
  - If the binding does *not return* an *AlreadyBoundException*, proceed to re-create the PersistentEventGroup and then remove the temporary binding. (Otherwise, another proxy is re-creating it. So wait for its completion.)

### 2.3. Garbage Collection

Both the Event Manager and Event Group objects must be prepared to garbage collect stale references. This is simply performed by deleting any objects (i.e., EventPost stubs) that it cannot communicate with. Hence their postEvent() methods handle garbage collection after notifying all interested objects of the event.

## 3. Changes to Agent Implementation

### CollaboratorAgent

The *CollaboratorAgent* class contains a reference to an event queue (i.e., an *EventQueueImpl* object), which is instantiated by its constructors.

The Event Queue does not persist across an Agent's travels. Hence, its *prepareForTransit()* method flushes the queue and voids the Agent's reference to the queue before it travels. Its associated Agent Groups save any events that occur while the Agent is in-transit. The *completedTransit()* method, called after the Agent arrives at its destination, constructs a new *EventQueueImpl* object and flushes any events saved by the associated Agent Groups while the Agent was in-transit.

### AgentStatus

The *AgentStatus* class contains a reference to the *EventPost* interface implemented by the CollaboratorAgent's Event Queue. It also contains an Agent's unique ID and current status. Hence, it will serve as both an Event Queue and a status indicator.

## Mitsubishi Electric ITA Confidential

**AgentGroupImpl**

The *AgentGroupImpl* class derives from *PersistentEventGroupImpl*. Thus, it not only enables Agent collaboration, but also provides reliable group event handling.

As a result, Agent Groups may implement different levels of persistence: Group membership is always written to persistent storage, but queued events for in-transit agents will also be saved if the Agent Group's configuration allows it.

Occasionally, the Agent Group may generate events that it directly forwards to a member of the group (e.g., to "ping" it after it fails to arrive at a collaboration point) or others that it "multicasts" to its members (e.g., an Agent's failure to reply to a ping). Group members may also broadcast an event to the group (e.g., an Agent may inform the group that it has encountered an exceptional condition).

**AgentEvent**

The *AgentEvent* class is the superclass of all events generated by Agents. It includes the Agent's unique ID.

```

AgentGroup.java_1 Mon Nov 11 16:25:52 1996 1
1  /*
2   * Header: /com/mitsubishi/zenezagents/collaboration/AgentGroup.java 5 16/01/96 3:40p Noemi S
3   * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4   * Information Technology Center America.
5   * All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC LTD.
8   */
9   * DESCRIPTION
10  */
11  */
12  */
13  */
14  * 5 10/01/96 3:40p Noemi
15  * Added some javaDoc comments
16  */
17  * 6 9/10/96 5:07p Noemi
18  * Changed return value of collaborate method from a void to an object.
19  */
20  * 3 9/06/96 3:54p Noemi
21  * Added UpdateAgent method. Changed argument to addAgents.
22  */
23  * 2 8/30/96 4:54p Noemi
24  * Added addAgent, removeAgent, and setGroupSize methods
25  */
26  * 1 8/23/96 2:13p Noemi
27  * Basic AgentGroup functionality
28  * mobility yet.
29  */
30  package com.mitsubishi.zenezagents.collaboration;
31  import java.util.List;
32  import java.util.concurrent.ExecutionException;
33  import java.rmi.Remote;
34  import java.rmi.RemoteException;
35  import java.rmi.RemoteInvocation;
36  import java.rmi.RemoteException;
37  */
38  /**
39  * AgentGroup is an RMI distributed interface. The actual
40  * implementation of AgentGroup is handled by AgentGroupImpl.
41  * This class provides RMI glue that enables Agents to invoke
42  * (remote) operations on an associate agent group.
43  */
44  */
45  * Agents may belong to one or more agent coordination groups
46  * (AgentGroups), which act as focal points for distributed
47  * synchronous agent collaboration and asynchronous notifications

```

```
48     * among members of the agent's group.
49     *
50     * @see Agent
51     * @see AgentGroupImpl
52     * @author Noemi Factoruk
53     */
54
55     public interface AgentGroup
56         extends Remote {
57
58     // Instance methods
59
60     /**
61      * Collaborator interface.
62      * @param result The result of the agent's computation.
63      * @exception RemoteException If an error occurs setting up network
64      * connections.
65      * @exception AgentGroupException If the collaboration times out.
66      * @return An object that describes the result of collaboration.
67
68     public Object collaborate(AgentResult result)
69         throws RemoteException, AgentGroupException;
70
71     /**
72      * Adds an Agent to the Group.
73      * @param status The AgentStatus object that describes the Agent.
74      * @exception RemoteException If an error occurs setting up network
75      * connections.
76
77     public void addAgent(AgentStatus status)
78         throws RemoteException;
79
80     /**
81      * Locates and removes the AgentStatus object containing the specified ID.
82      * @param agentID An Agent's unique ID.
83      * @exception RemoteException If an error occurs setting up network
84      * connections.
85      * @exception NoSuchElementException If the AgentStatus object does
86      * not exist.
87
88     public void removeAgent(String agentID)
89         throws RemoteException, ArrayIndexOutOfBoundsException, NoSuchElementException;
90
91     /**
92      * Replaces the AgentStatus object for the specified ID.
93      * This method may be used to change an Agent's status, ID, or both.
94      * @param agentID An Agent's unique ID
95      * @param newStatus An AgentStatus object to replace the existing one.
```

- 186 -

```

AgentGroup.java_1    Mon Nov 11 16:25:52 1996   3

95      * @exception RemoteException If an error occurs setting up network
96      * connections.
97      * @exception NoSuchElementException If the AgentStatus object does
98      * not exist.
99      */
100     public void updateAgent(String agentID, AgentStatus status)
101        throws RemoteException, ArrayIndexOutOfBoundsException, NoSuchElementException;
102
103     /**
104      * Returns the number of Agents in the group.
105      * @return The number of Agents in the group.
106      * @exception RemoteException If an error occurs setting up network
107      * connections.
108      */
109     public int getGroupSize()
110        throws RemoteException;
111
112     /**
113      * TURN ON LATER
114      *
115      */
116     public void agentException(String agentID, Exception type) throws RemoteException;
117
118     public void agentTermination(String agentID) throws RemoteException;
119
120     /**
121
122     */
123

```

- 187 -

```

AgentGroupException.java_  Non Nov 11 16:25:52 1996  1
1  /* $Header: /com/mitsubishi/nemesagents/collaborate/AgentGroupException.java 2   10/01/96 3:41p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION:
10 * Agent Group exceptions.
11 * $Log: AgentGroupException.java,v $
12 * Revision 1.1  1996-01-16 14:41:46 Noemi
13 * Added a protected default constructor and some javadoc comments.
14 * Revision 1.0  1996-01-16 14:41:46 Noemi
15 * AgentGroup related exceptions.
16 */
17 /**
18 * AgentGroupException defines exceptions that occur during
19 * package com.mitsubishi.nemesagents.collaborate;
20 import com.mitsubishi.nemesagents.shared.Agent;
21 */
22 /**
23 * AgentGroupException defines exceptions that occur during
24 * AgentGroup operations.
25 */
26 /**
27 * AgentGroupException defines exceptions that occur during
28 * AgentGroup operations.
29 */
30 /**
31 * Author: Noemi Pachon
32 */
33 public class AgentGroupException extends Exception {
34 /**
35 * Constructors
36 */
37 /**
38 * Constructs an AgentGroupException.
39 * @exception IllegalAccessException whenever called.
40 */
41 protected AgentGroupException() {
42     throws IllegalAccessException;
43 }
44 System.out.println("Default constructor called for AgentGroupException");
45 throw new IllegalAccessException("AgentGroupException Default constructor");
46 }
47

```

AgentGroupException.java\_2  
Mon Nov 11 16:25:52 1996

```
/*
 * Constructs an AgentGroupException
 * @param type A string describing the exception
 */
public AgentGroupException(String type) {
    super(type);
}
```

## AgentGroupImpl.java\_1 Mon Nov 11 16:25:52 1996 1

```

1  /*
2   * Slender: /com/mitsubishi/zones/agents/collaborate/AgentGroupImpl.java 7 10/01/96 3:43P Noemi S
3   *
4   * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
5   * Information Technology Center America.
6   * All rights reserved.
7   */
8   * Thread currentThread().getState() OF MITSUBISHI ELECTRIC LTD
9   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC LTD
10  * DESCRIPTION
11  * Agent Group distributed class.
12  * Log: /com/mitsubishi/zones/agents/collaborate/AgentGroupImpl.java 8
13  */
14  * 7 10/01/96 2:43P Noemi
15  * Added JavaDoc for thread()
16  * 6 9/20/96 12:37P Noemi
17  * Added agentsToString() method and some temporary logging to
18  * collaborate()
19  */
20  */
21  * 5 9/10/96 5:08P Noemi
22  * Changed code to use local and genericable methods for AgentGroup
23  */
24  * 4 9/06/96 3:57P Noemi
25  * Added remoteAgent, updatable, and indexAgent methods
26  * and agent's argument to the AgentGroup object.
27  */
28  * 3 9/04/96 3:33P Noemi
29  * Added getAgents() method.
30  */
31  * 2 8/30/96 4:25P Noemi
32  * Turned on code in constructor and finalizer that registers and
33  * deregisters with the RMI name server.
34  */
35  */
36  * 1 8/28/96 2:14P Noemi
37  * Basic synchronous agent collaboration functionality. Be support for
38  * remote groups, agent mobility, or asynchronous notifications.
39  */
40  */
41 package com.mitsubishi.zones.agents.collaborate;
42
43 import java.util.*;
44 import java.rmi.*;
45
46 import java.rmi.server.UnicastRemoteServer;
47 import java.rmi.RemoteException;

```

- 190 -

```

AgentGroupImpl.java_1    Mon Nov 11 16:25:52 1996   2

48 import java.rmi.Naming;
49 import java.rmi.RMISecurityManager;
50 import java.net.InetAddress;
51
52 /**
53 * AgentGroupImpl is an RMI distributed object. This is a base class
54 * that implements the AgentGroup interfaces.
55 *
56 * Agents may belong to one or more agent coordination groups
57 * (AgentGroups), which act as focal points for distributed
58 * synchronous agent collaboration and asynchronous notifications
59 * (e.g., exceptions) among members of the agent group.
60 *
61 * @see Agent
62 * @see AgentGroup
63 * @author Noemi Patocchi
64 * @author Noemi Patocchi
65 */
66
67 abstract public class AgentGroupImpl
68 extends UnicastRemoteObject
69 implements AgentGroup {
70
71 /**
72 * Default timeout for collaboration in ms.
73 */
74 public static final long DEFUALT_TIMEOUT = 100000;
75 /**
76 * Public static final long DEFUALT_TIMEOUT = 5000;
77 */
78
79 /**
80 * Protected string that get and ID = 0;
81 */
82 /**
83 * Instance variables
84 * protected long timeout;           // collaboration timeout in ms
85 * protected String groupName;        // name to register with RMI Name Server
86 * protected Vector agentList;        // Agent IDs of group members
87 * protected Vector resultList;       // results shared via collaborate
88 * protected boolean collaborationInProgress;
89 */
90
91 /**
92 * Constructs an AgentGroupImpl object.
93 */
94

```

- 191 -

```

AgentGroupImpl.java_1    Mon Nov 11 16:25:52 1996      3

95   public AgentGroupImpl();
96     throws RemoteException {
97   }
98
99
100  /**
101   * Constructs an AgentGroupImpl object.
102   * @param timeout The collaboration timeout in milliseconds
103   */
104
105  public AgentGroupImpl(long timeout)
106    throws RemoteException {
107
108    if (timeout < 0)    timeout = DEFAULT_TIMEOUT;
109
110    agentList = new Vector();
111
112    resultList = null;
113    ccilabrateOrInProgress = false;
114
115    /*
116     * Create a name to represent this instance of an agent group
117     * and register it with the RMI name service.
118     */
119    groupName = "AgentGroup" + nextGroupID();
120
121    try {
122      Naming.rebind(groupName, this);
123    } catch (Exception e) {
124      System.out.println("AgentGroupImpl: can't rebind: " + e.getMessage());
125      e.printStackTrace();
126    }
127
128
129    // Finalizer
130    protected void finalize()
131      throws Throwable {
132
133      System.out.println("AgentGroupImpl Finalizer called");
134
135      // Remove this Agent's Name from the RMI registry.
136      try {
137        Naming.unbind(groupName);
138      } catch (Exception e) {
139        System.out.println("AgentGroupImpl can't unbind: " + e.getMessage());
140        e.printStackTrace();
141

```

- 192 -

```

AgentGroupImpl.java_1      Mon Nov 11 16:25:52 1996   4

142
143      super.initialize();
144
145 }
146
147 // Class methods
148 private final synchronized int nextGroupID() {
149     return agentGroupID;
150 }
151
152
153 // Instance methods
154 /**
155 * Returns the Agent Group's unique ID.
156 * @return the Agent Group's unique ID.
157 */
158 public final String getGroupName() {
159     return groupName;
160 }
161
162
163 /**
164 * Converts an Agent Group into its string representation.
165 * @param status The AgentStatus object that describes the Agent Group
166 * @return A string representing the Agent Group
167 */
168 public String toString() {
169     return groupName;
170 }
171
172 // Methods to manipulate agentList.
173
174 /**
175 * Adds an Agent to the agentList.
176 * @param status The AgentStatus object that describes the Agent.
177 */
178 public synchronized void addAgent(AgentStatus status) {
179     agentList.addElement(status);
180     System.out.println("Added Agent, ID = " + status.getAgentID() +
181         ", Status = " + status.getStatus() + " to group");
182 }
183
184 /**
185 * Locates and removes the AgentStatus object containing the specified ID.
186 * @param agentID An Agent's unique ID.
187 * @exception NoSuchElementException If the AgentStatus object does
188

```

- 193 -

## AgentGroupImpl.java\_1 Mon Nov 11 16:25:52 1996 5

```

189      * not exist.
190      */
191      public synchronized void removeAgent(String agentID)
192          throws NoSuchElementException, ArrayIndexOutOfBoundsException {
193
194          int index = indexOfAgent(agentID);
195
196          if (index <= 1)
197              throw new NoSuchElementException("Can't find Agent ID!");
198          else {
199              AgentList removeElement (index);
200
201              // replaces the AgentStatus object for the specified ID.
202              // This method may be used to change an Agent's status, ID, or both.
203              // param Agent's unique ID
204              // param newstatus An AgentStatus object to replace the existing one.
205              // exception NoSuchElementException if the AgentStatus object does
206              // not exist.
207
208          }
209
210          public synchronized void updateAgent(String agentID, AgentStatus newStatus)
211              throws NoSuchElementException, ArrayIndexOutOfBoundsException {
212
213              removeAgent(agentID);
214              addAgent(newStatus);
215
216          }
217
218          // walks the AgentList looking for an AgentStatus object with the
219          // specified ID.
220
221          private int indexOfAgent(String agentID) {
222
223              int index, numAgents;
224              for (index = 0, numAgents = agentList.size(); index < numAgents; index++) {
225                  AgentStatus status = (AgentStatus)agentList.elementAt(index);
226                  if (agentID.compareTo(status.getAgentID()) == 0)
227                      return index;
228
229          }
230
231
232          /**
233          * Determines how many agents in the group have the specified status.
234          * @param status The status value to search for
235          * @return The number of Agent's with the specified status.

```

- 194 -

AgentGroupImpl.java\_1 Mon Nov 11 16:25:52 1996 6

```

236     * @see AgentStatus
237     */
238     public int agentsWithStatus(int status) {
239         int agentCount = 0;
240         for (Enumeration e = agentList.elements(); e.hasMoreElements(); ) {
241             AgentStatus aStatus = (AgentStatus)e.nextElement();
242             if (aStatus.getStatus() == status)
243                 agentCount++;
244         }
245         return agentCount;
246     }
247
248     /**
249      * Returns the number of agents in the group.
250      * Returns the number of agents in the group.
251      */
252     public final int getSize() {
253         return agentList.size();
254     }
255
256
257     /**
258      * Initiates the collaboration
259      */
260     /**
261      * Collaborator is the public interface to collaboration. It calls
262      * beginCollaboration() to synchronize the receiving of agent results
263      * and to wait for all agents to arrive.
264      *
265      * If all agents in the group have arrived at the collaboration
266      * point, each agent calls the application-specific analyzeResults()
267      * method to determine the agent's next course of action.
268      *
269      * Otherwise, the agent that initiated the collaborate calls
270      * receiverCollaborator()
271      *
272      * @param result The result of the agent's collaboration.
273      * @exception IOException If an error occurs setting up network
274      * connections.
275      * @exception AgentGroupException If the collaboration times out.
276      * @exception An Object that describes the result of collaboration
277      */
278     public Object collaborate(AgentResult result)
279     throws RemoteException, AgentGroupException {
280
281     System.out.println("Collaborate RMF");
282

```

- 195 -

```

AgentGroupImpl.java_1      Mon Nov 11 16:25:52 1996   7

283     boolean firstCollaborator = beginCollaboration(result);
285
286     /*
287      * If collaboration failed and this is the agent that initiated
288      * collaboration, it woke up because its timer expired. It retries
289      * once before waking up the other agents.
290
291     */
292     if (firstCollaborator && !collaborationSucceeded()) {
293         System.out.println("Collaboration Failed. Retrying ...");
294         synchronized(mutex) {
295             try {
296                 this.wait(timeout);
297             } catch (InterruptedException e) {
298             }
299         }
300     }
301     /*
302      * If all agents in the group have arrived at the collaboration
303      * point, each agent calls the application-specific analyzeResults()
304      * method to determine the agent's next course of action.
305
306     */
307     if (collaborationSucceeded()) {
308         System.out.println("Ready to do collaboration.");
309         Object collaboration = analyzeResults(result);
310
311         /*
312          * The agent that originated collaboration cleans up
313          * its firstCollaborator
314          * end collaboration();
315
316         */
317         if (firstCollaborator) {
318             beginCollaborator();
319             endCollaboration();
320
321             return collaboration;
322
323         }
324         if (firstCollaborator) {
325             System.out.println("Collaboration Failed while " +
326             agentStatus(AGENTSTATUS.INTRANSIT) +
327             " agents were in transit");
328
329         synchronized(this) {

```

- 196 -

8

AgentGroupImpl.java\_1

Mon Nov 11 16:25:52 1996

```

330     }
321     recoverCollaboration();
322   } else {
323     System.out.println("Collaboration failed");
324   }
325   throw new AgentGroupException("Collaboration timed out");
326 }
327 }
328 }
329 }
330 }

340
341
342 /**
343 * beginCollaboration() is called internally by collaborate() to
344 * synchronize each agent's recording of results and to wait for
345 * all agents to arrive at the collaboration point. This method acts
346 * as a distributed rendezvous, cf sorts. Each caller blocks until
347 * all agents in the group have "arrived" at the collaboration point.
348 * This method returns a boolean that indicates if the agent initiated
349 * the collaboration.
350 */
351 * @param result The result of the agent's computation
352 * @exception AgentGroupException If the group only has one member.
353 * @return A boolean indicating if the collaboration was successful.
354 */
355

356 /**
357 * protected synchronized boolean beginCollaboration(AgentGroupImpl result)
358 * throws AgentGroupException {
359 * boolean expectedResult = false;
360 *
361 * System.out.println("Beginning collaboration");
362 *
363 * if (collaboratingProgress) {
364 * /*
365 * If this agent is initiating the collaboration, allocate the
366 * resultList Vector and add result to it. Then wait for all
367 * agents to arrive at the collaboration point or for the
368 * collaboration timer to expire.
369 * */
370 * if (agentList.size() < 2) {
371 * endCollaboration();
372 * throw new AgentGroupException("AgentGroup too small");
373 * }
374 * collaborationProgress = true;
375 }


```

- 197 -

```

AgentGroupImpl.java_1   Mon Nov 11 16:25:52 1996      9

377     firstCollaboration = true;
378     resultList = new Vector();
379     resultList.addElement(result);
380
381     try {
382       this.waitTimeSet;
383       if (e instanceof InterruptedException) {
384         } else {
385           /*
386            * Add this agent's results to resultList. If this is the
387            * last agent to arrive, wake up the other agents. Otherwise
388            * wait for the rest of the agents in the group to arrive.
389            */
390           resultList.addElement(result);
391           if (collaborationsServiceended())
392             this.notifyAll();
393           else {
394             /*
395              * This waiting
396              * action. If an InterruptedException e) {
397
398
399           }
400
401           return (Vector) result;
402         }
403
404
405         /*
406          * Clean up after a successful collaboration.
407          */
408         protected void endCollaboration() {
409           collaborations = false;
410         }
411
412
413         /**
414          * Clears up after a failed collaboration.
415          */
416         protected void recoverCollaboration() {
417           endCollaboration();
418
419
420         }
421
422
423        /**

```

AgentGroupImpl.java\_1 Mon Nov 11 16:25:52 1996 10

```

424     * analyzeResults() is an abstract method that must be overridden by
425     * each application. It provides the application-specific behavior
426     * required to implement collaboration. This method is invoked on
427     * behalf of each agent in the group. It is passed the results
428     * computed by each agent and determines the course of action for the
429     * calling agent, or any based on the computed results
430     *
431     * @param results An enumeration of the results computed by the agents
432     * @return An application-specific object or null
433     */
434     abstract protected void analyzeResults(Enumeration results);
435
436
437     /**
438     * Private method to determine if all agents have reached the
439     * collaboration point. This method is synchronized to prevent
440     * races with agents joining and leaving the group.
441     */
442     private synchronized boolean collaborationSucceeded() {
443         return (resultList.size() == agentList.size());
444     }
445
446
447     /**
448     * TURNS ON LATER
449     */
450
451     /**
452     * Methods to handle exceptions and other notifications
453     * provided by AgentGroup.
454     */
455
456     public void agentTermination(int agentID)
457         throws RemoteException {
458     }
459
460
461     /**
462     */
463

```

-155-

```

AgentResult.java_1      Mon Nov 11 16:25:52 1996      1
1  /* $Header: /com/mitscagents/collaborate/AgentResult.java 4   10/01/96 3:43p Noemi $
2
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America
5  * All rights reserved
6
7  * CONFIDENTIAL AND PROPERTY OF Mitsubishi ELECTRIC ITA
8
9  * DESCRIPTION
10 * This object represents the result of an agent's computation.
11 *
12 * $Log: AgentResult.java,v $
13 * Revision 1.1  10/01/96 3:43p Noemi
14 * Added some Javadoc comments
15 *
16 * Revision 1.0  9/06/96 3:45p Noemi
17 * Added a protected default constructor to ensure that the public
18 * constructor is used for creating variables correctly.
19 *
20 * Revision 1.0  8/26/96 2:17p Noemi
21 * Changed private protected instance variables to protected.
22 *
23 * Revision 1.0  8/26/96 2:17p Noemi
24 * A generic class that represents the result of an agent computation.
25 * (Used by collabagents module)
26 *
27 */
28 package com.mitscagents.collaborate;
29
30
31 /**
32 * AgentResult contains an agent ID and a result to be shared
33 * via collaboration
34 *
35 * @see CollaborateAgent
36 * @see AgentResultImpl
37 * @author Noemi Panjwani
38 *
39 */
40 public class AgentResult {
41
42     // Instance variables
43     protected String agentID;           // ID of agent that computed result
44     protected Object result;           // actual result computed
45
46     // Constructors
47
}

```

- 260 -

```

AgentResult.java_1   Sun Nov 12 16:25:52 1996    2

48     * Protected Default Constructor for an AgentResult object.
49     * @exception IllegalAccessException Whenever called.
50     */
51 protected AgentResult()
52 {
53     this();
54 }
55
56
57 /**
58  * Constructs an AgentResult object.
59  * @param agentID the Agent's unique ID.
60  * @param result the result to be shared via collaboration.
61  */
62 public AgentResult(String agentID, Object result)
63 {
64     this.agentID = agentID;
65     this.result = result;
66 }
67
68 /**
69  * Instance method
70  */
71 /**
72  * Sets an Agent's unique ID from this AgentResult object.
73  * @param agentID the Agent's unique ID.
74  */
75 public final String setAgentID()
76 {
77 }
78
79 /**
80  * Sets the result to be shared via collaboration
81  * from this AgentResult object.
82  * @param result the result to be shared via collaboration.
83  */
84 public Object getResult()
85 {
86 }
87

```

*- 201 -*

```

AgentStatus.java_1           Mon Nov 11 16:25:53 1996      1
1  /* $Header: /com/mitsubishi/zonestagents/collaborate/AgentStatus.java 2   10/01/96 3:44p Noemi $
2  *
3  * Copyright 1996 Mitsubishi Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC USA.
8  *
9  * DESCRIPTION
10 * This object holds an Agent's status (e.g. ACTIVE).
11 *
12 * $Log: /com/mitsubishi/zonestagents/collaborate/AgentStatus.java $
13 * Revision 1.2          10/01/96 3:44p Noemi
14 * Added some javadoc comments.
15 *
16 * Revision 1.1          9/06/95 3:45p Noemi
17 * This class represents an agent's current status (e.g., ACTIVE).
18 *
19 *
20 package com.mitsubishi.zonestagents.collaborate;
21
22 /**
23 * AgentStatus contains an agent ID and a status value. Each AgentGroup
24 * contains an AgentStatus object for each Agent in the group.
25 *
26 * @see Collaborate
27 * @see AgentGroup
28 * @author Noemi Parikh
29 */
30
31 public class AgentStatus {
32
33     /**
34     * Constraints
35     * Private static final int FIRST_STATUS = 1;
36     */
37     * Agent is active.
38     */
39     public static final int ACTIVE = 1;
40     */
41     * Agent is inactive.
42     */
43     public static final int INACTIVE = 2;
44     */
45     * Agent is in an unknown state.
46     */
47     public static final int UNKNOWN = 3;

```

- 202 -

```

AgentStatus.java_1      Mon Nov 11 15:25:53 1996   2

48    private static final int LAST_STATUS = UNKNOWN;
49
50    // Instance variables
51    protected String agentID;    // ID of agent
52    int currentStatus;
53
54    // Constructors
55    /**
56     * Constructs an AgentStatus object.
57     * @param description    Whenever called.
58     */
59    protected AgentStatus() {
60        throws IllegalAccessException;
61        System.out.println("Default constructor called for AgentStatus");
62        throw new IllegalAccessException("AgentStatus default constructor");
63    }
64
65
66    /**
67     * Constructs an AgentStatus object.
68     * @param agentID    The agent's unique ID.
69     * @param status    The agent's status
70     */
71    public AgentStatus(int agentID) {
72        this(agentID, ACTIVE);
73    }
74
75    /**
76     * Constructs an AgentStatus object.
77     * @param agentID    The agent's unique ID.
78     * @param status    The agent's status
79     */
80    public AgentStatus(int agentID, int status) {
81        this.agentID = agentID;
82        if (status < FIRST_STATUS || status > LAST_STATUS)
83            UNKNOWN = status;
84    }
85
86    /**
87     * Instances return.
88     */
89    /**
90     * Extracts an Agent's unique ID from an AgentStatus object.
91     * @return An Agent's unique ID.
92     */
93    public final int getID() { return 100; }
94

```

- 203 -

AgentStatus.java\_1 Mon Nov 11 16:25:53 1996 3

```
95
96
97     /**
98      * Extracts an Agent's current status from an AgentStatus object.
99      * @return An Agent's current status
100     */
101    public final int getStatus() {
102        return status;
103    }
104
105 }
```

- 6 -

```

CollaboratorAgent.java_1      Mon Nov 11 16:25:53 1996      1
1  /*
2   *  $Header: /com/meitaca/hall/zenesagents/collaborate/CcollaboratorAgent.java 5  10/01/96 3:45p Noemi $
3   *  Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *  All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Performed class transformation AgentCollaboration
11  *  Signs: AgentCollaboration AgentCollaboration
12  *
13  *  $Log: CcollaboratorAgent.java,v $
14  *  5 10/01/96 3:45p Noemi
15  *  Added some JavaDoc comments
16  *  4 9/20/96 4:10p Noemi
17  *  Changed finalizer to ensure that it performs cleanup no more than once.
18  *  (A bug in the package collaborator may cause the finalizer to be called
19  *  more than once)
20  *
21  *  3 9 10/96 5:39p Noemi
22  *  Renamed setIntransitStatus and setActiveStatus to prepareForTransmit
23  *  and completeTransmit, respectively.
24  *
25  *  2 9/05/96 3:57p Noemi
26  *  Added the following methods: addGroup, removeGroup, intransitStatus,
27  *  setActiveStatus, and completeTransmit.
28  *
29  *  1 9/04/96 3:15p Noemi
30  *  Subclass of Agent that implements a Collaborating Agent.
31  *
32  */
33
34 package com.meitaca.hall.zenesagents.collaborate;
35
36 import java.util.*;
37 import java.net.*;
38
39 import com.meitaca.hall.zenesagents.shared.Agent;
40
41 /**
42  * A subclass of Agent that enables collaboration among agents in a group.
43  * Agents wishing to collaborate must extend this class to do some useful work.
44  * They must also belong to an agent coordination group (AgentGroup). Generally,
45  * an application first creates an AgentGroup. Then it obtains a reference to
46  * the RMI interface for the group by looking up the group in the local RMI name
47  * service. Subsequently, it creates new CollaboratorAgents and populates the

```

- 205 -

```

CollaboratorAgent.java_1  Mon Nov 11 16:25:53 1996   2

48     * group with them (by passing the constructor a reference to the interface).
49     *
50     * @see Agent
51     * @see AgentGroup
52     * @author Iremi Farzadk
53     */
54     public abstract class CollaboratorAgent
55     extends Agent
56
57     /**
58      * Instantiates variables
59      * protected vector groups;
60      */
61
62     /**
63      * Constructs a CollaboratorAgent
64      * @param id A unique identifier
65      * @param groups A list of AgentGroups
66      */
67     protected CollaboratorAgent()
68     throws IllegalArgumentException {
69
70     System.out.println("Default constructor for CollaboratorAgent()");
71
72     /**
73      * Constructs a CollaboratorAgent
74      * @param id A unique identifier
75      * @param groups A list of AgentGroups to which this Agent belongs
76      */
77     public CollaboratorAgent(String id, Vector groups) {
78
79     /**
80      * Set the agentID in the AgentGroup and save it
81      * to the list of the AgentGroup in groups.
82
83     groupList = new Vector();
84
85     /**
86      * Constructs a CollaboratorAgent
87      * @param groups A list of AgentGroups to which this Agent belongs
88      */
89     public CollaboratorAgent(Vector groups) {
90
91     this(groups, null);
92
93
94
}

```

```

95     /**
96      * Constructs a CollaboratorAgent
97      * @param groups A list of AgentGroups to which this Agent belongs.
98      * @param agentID the unique ID of an agent to be replaced by this agent.
99      */
100    public CollaboratorAgent(AgentGroup groups, String agentID) {
101
102      /*
103       * Save a copy of agentID in each of the dissociated AgentGroups and
104       * store a reference to each AgentGroup in groupList.
105       */
106    groupList = new Vector();
107
108    for (Enumeration e = groups.elements(); e.hasMoreElements(); ) {
109      AgentGroup group = (AgentGroup)e.nextElement();
110
111      try {
112        if (agentID == null)
113          group.removeAgent(new AgentStatus(agentID));
114
115        group.updateAgent(agentID, new AgentStatus(agentID));
116
117        System.out.println("Group size = " + group.getGroupSize());
118
119        catch (Exception ex) {
120          System.out.println("Agent construction failed agent to group: " +
121              agentID);
122          System.out.println("AgentID = " + agentID + ", oldID = " + oldID);
123          ex.printStackTrace();
124
125        }
126
127        /*
128         * Finalize
129         */
130        protected void finalize() {
131          throws Throwable {
132
133          /*
134           * Remove agentID from all groups this agent belongs to.
135           */
136          if (groupList != null) {
137            for (Enumeration e = groupList.elements(); e.hasMoreElements(); ) {
138              AgentGroup group = (AgentGroup)e.nextElement();
139              group.removeAgent(agentID);
140
141            } catch (Exception ex) {

```

- 207 -

Mon Nov 11 16:25:53 1996 4

```

CollaboratorAgent.java_1
System.out.println("CollaboratorAgent finalizer: can't remove agent from group: " +
    ex.getMessage());
ex.printStackTrace();
}

}
groupList = null; // in case finalizer gets called twice (it has been!)

}
catch(FinalizerException e) {
    System.out.println("FinalizerException caught");
}

// Instance methods
...
* Returns a list of groups the CollaboratorAgent belongs to.
* @return the list of groups this CollaboratorAgent belongs to.
*/
156 public final synchronized Enumeration getGroups() {
157     return groupList.elements();
158 }
159 }

160 /**
* Add a group to this agent's groupList and add the agent
* to the group's agent list.
* @param group The group to be added.
*/
161 public synchronized void addGroup(AgentGroup group) {
162     groupList.addElement(group);
163 }
164 }

165 /**
* Remove a group from this agent's groupList and remove the agent
* from the group's agent list.
* @param group The group to be removed.
*/
166 public synchronized void removeGroup(AgentGroup group) {
167     groupList.removeElement(group);
168 }
169 }

170 /**
* System.out.println("Group size = " + group.size());
* @param ex Exception
*/
171 public void printIn(Enumeration group) {
172     System.out.println("Can't add agent to group: " + ex.getMessage());
173     ex.printStackTrace();
174 }
175 }

176 }

177 /**
* Remove a group from this agent's groupList and remove the agent
* from the group's agent list.
* @param group The group to be removed.
*/
178 public synchronized void removeGroup(AgentGroup group) {
179     groupList.removeElement(group);
180 }
181 }

182 /**
* System.out.println("Can't remove agent: " + ex.getMessage());
* @param ex Exception
*/
183 public void removeAgent(AgentID agentID) {
184     try {
185         group.removeAgent(agentID);
186     } catch( Exception ex ) {
187     }
188 }

```

- 208 -

```

CollaboratorAgent.java_1    Mon Nov 11 16:25:53 1996   5
                           System.out.println("Can't remove agent from group: " + ex.getMessage());
                           ex.printStackTrace();
}
}

194      /**
195       * Update an agent's status in all groups it belongs to.
196       * @param status The new status.
197
198      */
199      public synchronized void updateStatus(int status) {
200          if (groupList != null) {
201              for (Enumeration e = groupList.elements(); e.hasMoreElements(); ) {
202                  AgentGroup group = (AgentGroup)e.nextElement();
203                  try {
204                      group.updateAgent(agentID, new AgentStatus(agentID, status));
205                  } catch (Exception ex) {
206                      System.out.println("CollaboratorAgent finalizer: can't remove agent from group: " +
207                                      ex.getMessage());
208                      ex.printStackTrace();
209                  }
210
211
212      /**
213       * Set an agent's status to INTRANSIT before transport.
214       */
215      public void prepareForTransport() {
216          updateStatus(AgentStatus.INTRANSIT);
217      }
218
219
220      /**
221       * Set an agent's status to ACTIVE after transport.
222
223      */
224      public void completedTransport() {
225          updateStatus(AgentStatus.ACTIVE);
226      }
227
228  }

```

- 209 -

EventException.java\_1      Mon Nov 11 16:26:41 1996      1

```

1  /* $Header: /com/mitsc/hs1/zones/agents/event/EventException.java 1      10/14/96 5:55p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 */
11 */
12 */
13 */
14 */
15 */
16 */
17 */
18 */
19 */
20 */
21 */
22 */
23 */
24 */
25 */
26 */
27 */
28 */
29 */
30 */
31 */
32 */
33 */
34 */
35 */
36 */
37 */
38 */
39 */
40 */
41 */
42 */
43 */
44 */
45 */
46 */
47 */

```

SLUG: com.mitsc.hsi.zones.agents.event.EventException.java \$

1 10/14/96 5:55p Noemi

1 Base class for exceptions generated during event processing.

1 package com.mitsc.hsi.zones.agents.event;

1 \* Exceptions encountered by Events.

1 \* @see EventType

1 \* @author Noemi Noemi

1 public class EventException extends Exception {

1 \* Constructors

1 \* Constructors for EventException

1 \* \* exception: the original exception whenever null:

1 \* \* protected EventException(

1 \* throws IOException, PrinterException,

1 \* System.out.println("Default constructor called for EventException");

1 \* \* new StackAccessException(EventException.super(EventException)));

1 \* }

1 \* Constructors for printing:

1 \* \* param type: a string describing the exception

1 \* \* /

1 \* public EventException(String type) {

1 \* super(type);

1 \* }

## EventGroup.java\_1      Mon Nov 11 16:26:41 1996      1

```

1  /* $Header: /com/trizetta/zc_zesagents/event/EventGroup.java 1 10/14/96 6:59p Noemi $
2  *
3  * Copyright 1991 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * Event Group interface file.
11 */
12 * $Log: /com/trizetta/zesagents/event/EventGroup.java $
13 * 1 10/14/96 6:59p Noemi
14 * Event Group Interface.
15 */
16 */
17 */
18 package com.trizetta.zesagents.event;
19 import com.trizetta.zesagents.event.Event;
20 import java.util.List;
21 import java.util.concurrent.ExecutionException;
22 import java.rmi.Remote;
23 import java.rmi.RemoteException;
24 import java.rmi.RemoteException;
25 */
26 * EventGroup is a distributed RMI interface for event groups.
27 * An EventGroup instance is a group of objects that wish to receive
28 * events generated by the other members of the group. The EventGroup
29 * serves as a gateway between the objects in the group and forwards
30 * events posted to it by the members of the group.
31 * The members of the group must all implement the EventPost interface.
32 */
33 * @see EventGroupImpl
34 * @author Noemi Farcash
35 */
36 */
37 */
38 */
39 public interface EventGroup
40 extends Remote {
41 */
42 */
43 */
44 * Adds a member to the event group.
45 * @param object An object that implements the EventPost interface.
46 * @exception RemoteException If an error occurs setting up network
47 * connections

```

EventGroup.java\_1      Mon Nov 11 16:26:41 1996      2

```
48        * @exception ClassCastException If object does not implement EventPost.
49        */
50        public void addMember(CObject object) {
51           throws RemotingException, ClassCastException;
52        }
53        /**
54        * Removes a member from the group.
55        * @param object The object to remove
56        * @exception RemotingException If an error occurs setting up network
57        * connections
58        * @exception NoSuchElementException If object is not a member of the group.
59        */
60        public void removeMember() throws RemotingException, NoSuchElementException;
61        throws RemotingException, NoSuchElementException;
62        }
63 }
```

- 212 -

```

EventGroupImpl.java_1    Mon Nov 11 16:26:41 1996      1
1  /*  SHeader: /com/meitca/hsl/zonesagents/event/EventGroupImpl.java 2   10/22/96 7:10p Noemi $ 
2  *
3  * Copyright 1995 Horizon Systems Laboratory. Mitsubishi Electric
4  * Information Technology Center of America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  *
9  * DESIGNATION:
10 * Event Group distributed object.
11 * $Log: /com/meitca/zonesagents/event/EventGroupImpl.java $
12 *
13 * 2 10/22/96 7:10p Noemi
14 * Changed the class cause of the postEvent method.
15 *
16 * 1 10/14/96 1:59p Noemi
17 * Event group implementation.
18 *
19 *
20 *
21 package com.meitca.zonesagents.event;
22
23
24 import java.util.*;
25 import java.io.*;
26
27 import java.rmi.server.UnicastRemoteServer;
28 import java.rmi.RemoteException;
29 import java.rmi.Naming;
30 import java.rmi.AlreadyBoundException;
31
32 import com.meitca.zonesagents.Event;
33
34 /**
35 * EventGroupImpl is an RMI server, created object that implements
36 * Event Group and EventPost.
37 *
38 *
39 * An EventGroup consists of a group of objects that wish to receive
40 * events generated by the other members of the group. The EventGroup
41 * serves as a gateway between the objects in the group and forwards
42 * events posted from to the members of the group.
43 *
44 * The members of the group must all implement the EventPost interface.
45 *
46 * @see EventGroup
47 * @author Noemi Paci

```

- 213 -

```

EventGroupImpl.java_1    Mon Nov 11 16:26:41 1996   2

48   */
49   public class EventGroupImpl {
50     extends UnicastRemoteServer
51     implements EventGroup, EventPost {
52
53     // instance variables
54
55     /**
56      * A list of the group's members.
57      */
58     protected Vector memberList;
59
60     /**
61      * Constructs an EventGroupImpl object.
62      * @exception RemoteException if an error occurs creating up network
63      * connections.
64
65     public EventGroupImpl () throws RemoteException {
66
67       /**
68        * Create member list and event queue.
69        */
70       memberList = new Vector();
71
72       instanceLock = new Object();
73
74       System.out.println("EventGroupImpl created");
75
76       addMember(null);
77
78     /**
79      * Add a member to the event queue.
80      * @param member An object that implements the EventPost interface.
81      * @exception RemoteException If client does not implement EventPost.
82      */
83      synchronized void addMember(Object object) {
84
85        EventPost eventPost = (EventPost) object;
86
87        try {
88          eventPost.eventPost();
89        } catch (ClassCastException e) {
90          System.out.println("EventGroupImpl: Object does not implement Event Post");
91        }
92
93
94      /**

```

- 214 -

EventGroupImpl1.java\_1 MGN NCV 11 16:26:41 1996

```

95     * Remove a member from the group
96     * @param object the Object to remove
97     * @exception NoSuchElementException If object is not a member of the group.
98     */
99    public synchronized void removeMember(Object object)
100        throws NoSuchElementException {
101        if (!membersList.removeElement(object))
102            throw new NoSuchElementException("Object not member of group");
103    }
104
105    /**
106     * EventPost methods
107     */
108    /**
109     * Forward an event to all members of the group.
110     * @param event The event to post.
111     * @exception EventException If an error occurs posting the event.
112     */
113    public void postEvent(EventType event)
114        throws RemoteException, EventException, IOException {
115        for (Enumeration e = membersList.elements(); e.hasMoreElements(); ) {
116            EventPost eventQueue = (EventPost)e.nextElement();
117            eventQueue.postEvent(event);
118        }
119    }
120}
121
122}

```

- 215 -

```

EventHandler.java_1      Mon Nov 11 16:26:41 1996      1
1  /* $Header: /ccm/meetca/bsl/zonesjagents/event/EventHandler.java 1
2   *
3   * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4   * Information Technology Center America.
5   * All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   */
9   * DESCRIPTION
10  * Event handler interface
11  */
12  * $Log: /ccm/meetca/bsl/zonesjagents/event/EventHandler.java $
13  *
14  * Revision Log:
15  * 1 10/14/96 6:56p Noemi
16  * Event handler interface.
17  */
18 package com.melice.bsl.zonesjagents.event;
19
20 import java.rmi.Remote;
21 import java.rmi.RemoteException;
22
23 /**
24  * EventHandler is a distributed RMI interface for handling events.
25  *
26  * @author Noemi Parfrey
27  */
28 public interface EventHandler {
29
30     // Instance methods.
31
32     /**
33      * Event handler interface.
34      * @param event current event
35      * @exception RemoteException if an error occurs setting up network
36      * connections.
37      * @exception EventException if an error occurs handling the event.
38     */
39     public void handleEvent(EventType event)
40         throws RemoteException, EventException;
41
42 }

```

```

EventManager.java_1      Mon Nov 11 16:26:41 1996      1
1  /* $Header: /com/meitca/hsl/zonesagents/event/EventManager.java 2   10/22/96 7:12p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  /*
10 * DESCRIPTION
11 * Event Manager Interface.
12 * $Log: /com/meitca/hsl/zonesagents/event/EventManager.java $
13 */
14 * 2 10/22/96 7:12p Noemi
15 * Changed registerEvents and unregisterEvents to take an array of
16 * Strings, rather than an array of EventType. Also changed several of
17 * the throws clauses.
18 *
19 * 1 10/14/96 6:45p Noemi
20 * Event Manager interface.
21 */
22 package com.meitca.hsl.zonesagents.event;
23
24 import java.io.*;
25 import java.rmi.Remote;
26 import java.rmi.RemoteException;
27
28 /**
29 * EventManager is a distributed RMI interface for event management.
30 *
31 * @see EventManagerImpl
32 * @author Noemi Pachorek
33 */
34
35 public interface EventManager
36     extends Remote {
37
38 /**
39 * Instance methods.
40 * Methods used to register interest in specified events.
41 */
42 /**
43 * Register interest in receiving specified events.
44 * @param events An array of the names of events that an object
45 * is interested in receiving.
46 * @param eventQueue The associated event queue's EventPost interface.
47 * @exception RemoteException If an error occurs setting up network

```

- 217 -

```

48     * connections.
49   */
50   public void registerEvents(String events[], EventPost eventQueue)
51       throws RemoteException, IOException;
52
53 /**
54  * Register interest in receiving all events posted to this EventManager.
55  * @param eventQueue The associated event queue's EventPost interface.
56  * @exception RemoteException If an error occurs setting up network
57  * connections.
58  */
59   public void registerEvents(EventPost eventQueue)
60       throws RemoteException, IOException;
61
62 // Methods to unregister events.
63 /**
64  * Remove registration for specified events.
65  * @param events An array of the names of events that an object
66  * is no longer interested in receiving.
67  * @param eventQueue The associated event queue's EventPost interface.
68  * @exception RemoteException If an error occurs setting up network
69  * connections.
70  * @exception EventManagerException If the event queue was not registered
71  * for one or more of the specified exceptions.
72  */
73   public void unregisterEvents(String events[]) EventPost eventQueue)
74       throws RemoteException, IOException, EventManagerException;
75
76 /**
77  * Remove registration for all events.
78  * @param eventQueue The associated event queue's EventPost interface.
79  * @exception RemoteException If an error occurs setting up network
80  * connections.
81  */
82   public void unregisterAll(EventPost eventQueue)
83       throws RemoteException, IOException;
84
85 }

```

- 218 -

Mon Nov 11 16:26:41 1996 1

```

1  /* $Header: /com/meitaca/hsl/zonesagents/event/EventManagerException.java 1   10/14/96 6:57p Noemi $
2   *
3   * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4   * Information Technology Center America.
5   * All rights reserved.
6   */
7   * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITS.
8   */
9   * DESCRIPTION
10  * EventManager exception.
11  */
12  * $Log: /com/meitaca/hsl/zonesagents/event/EventManagerException.java $
13  * 1 10/14/96 6:57p Noemi
14  * Exceptions generated by the Event Manager.
15  */
16  package com.meitaca.hsl.zonesagents.event;
17
18  /**
19  * Exceptions encountered by the EventManager.
20  */
21  @see EventManagerImpl
22  * Author Noemi Pactorek
23  */
24
25 public class EventManagerException extends EventException {
26
27  /**
28  * Constructors an EventManagerException whenever called.
29  */
30  * Constructs an EventManagerException whenever called.
31  * @param type A string describing the exception.
32  */
33 protected EventManagerException()
34 throws IllegalAccessException {
35
36 System.out.println("Default constructor for EventManagerException");
37 throw new IllegalAccessException("EventManagerException Default constructor");
38
39
40 /**
41  * Constructs an EventManagerException.
42  * @param type A string describing the exception.
43  */
44 public EventManagerException(String type) {
45
46     super(type);
47 }

```

- 215 -

- 220 -

```

EventManagerImpl.java_1   Mon Nov 11 16:26:42 1996      1
1  /* $Header: /com/meitca/hsl/zonesagents/event/EventManagerImpl.java 3 10/24/96 4:57p Noemi $
2  *
3  * Copyright 1996 Herizen Systems Laboratory. Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * Event Manager distributed object.
11 */
12 * $Log: /com/meitca/hsl/zonesagents/event/EventManagerImpl.java $
13 */
14 * 3 10/24/96 4:57p Noemi
15 * Added support for an Event Manager properties file and properties that
16 * specify if persistence is enabled and the name of the persistent
17 * storage file.
18 */
19 * 2 10/22/96 7:12p Noemi
20 * Save full and specific event registrations to persistent storage.
21 * Constructor reads persistent data and reconstructs notifications.
22 * EventManager removes event queues that it can't post events to.
23 * Changed registerEvents and unregisterEvents to take an array of String.
24 * Made constructor protected and added a main method.
25 * Perform cleanup when receiving conflicting registrations (final vs.
26 * specific).
27 */
28 * 1 10/14/96 6:59p Noemi
29 * The Event Manager.
30 */
31 package com.meitca.hsl.zonesagents.event;
32
33 import java.util.*;
34 import java.rmi.*;
35 import java.io.*;
36
37 import java.rmi.server.UnicastRemoteServer;
38 import java.rmi.RemoteException;
39 import java.rmi.Naming;
40 import java.rmi.NotBoundException;
41 import java.rmi.AlreadyBoundException;
42
43 import com.meitca.hsl.zonesagents.shared.*;
44
45 import com.meitca.hsl.zonesagents.security.InsecurityManager;
46

```

```

48 /**
49 * EventManagerImpl is an Event Manager that implements the RMI
50 * interface and EventPost interfaces.
51 *
52 * This class allows objects to register their interest in receiving
53 * specified events and to delete their registrations.
54 *
55 * Objects post events by calling the EventManager's postEvent method
56 * (defined by the EventPost interface).
57 *
58 * The EventManager is a central object (i.e., there is only one)
59 * per system.
60 *
61 * @see EventManager
62 * @see EventPost
63 * @see EventHandle
64 * @see EventManagerProxy
65 * @author Noemi Fischer
66 */
67
68 public class EventManagerImpl
69 extends UnicastRemoteObject
70 implements EventManager, EventPost {
71
72 // Constants
73 private static final String EVENT_MANAGER_NAME = "EventManager";
74
75 private static final String PROPERTIES_DESCRIPTION = "Properties File for EventManager";
76 private static final String ENABLE_PERSISTENCE = "EventManager.Persistence.Enable";
77 private static final String PERSISTENCE_FILE = "EventManager.Persistence.FileName";
78 // Private static final String DEFAULT_PERSISTENCE_FILE = "C:\Temp\EventManager.store";
79 private static final String DEFAULT_PERSISTENCE_FILE = "EventManager.store";
80
81 /* IDs for persistence objects */
82 private static final int EVENT_REGISTRATION_ID = 1;
83 private static final int FULL_REGISTRATION_ID = 2;
84
85 private static final int STORAGE_PADDING = 1024;
86
87
88 // Instance variables
89
90
91 /**
92 * Hash table that maps class names derived from EventType to
93 * registered EventPost stubs.
94 */

```

```

EventManagerImpl.java_1   Mon Nov 11 16:26:42 1996   3

95     protected Hashtable eventRegistrations;
96
97     /** List of EventPost stubs registered to receive all notifications. */
98     protected Vector fullRegistrations;
99
100    /**
101     * Hash table that maps an EventPost stub to the list of events it
102     * is registered for (containing it maps the stub to the class name of
103     * the event).
104     */
105    protected Hashtable eventRegistrations;
106
107    /**
108     * A reference to this EventManager's persistent store handler.
109     */
110    protected PersistenceStorageHandler persistenceManager;
111
112    /**
113     * Name of the persistent storage file.
114     */
115    protected String persistentStorageFile;
116
117    /**
118     * Constructs an EventManager. EventManagerImpl has no public
119     * constructors. Applications wishing to utilize the Event Manager's
120     * services must access it via EventManagerProxy.
121     */
122    protected EventManagerImpl() {
123        throws Exception;
124
125        /**
126         * Locate properties file
127         */
128        try {
129            properties = new JASProperties(EVENT_MANAGER_NAME,
130                                         JAS_PROPERTIES_SERVER, PROPERTIES_DESCRIPTION, false);
131            System.out.println("EventManagerImpl: Located properties file");
132
133        } catch (Exception e) {
134            System.out.println("EventManagerImpl: Couldn't access Properties file or Conduit Ser
135            System.out.println("\tContinuing with defaults.");
136            // Construct an empty Properties object.
137            properties = new JASProperties();
138
139        }
140
141        /**
142         * Recover event and full registrations from persistent storage,

```

- 223 -

```

EventManagerImpl.java_1   Mon Nov 21 16:26:42 1996   4
142      * if required. Then reconstruct event notifications.
143
144      */
145      recoverRegistrations();
146
147      /**
148       * Register this instance with the RMI registry.
149       */
150      Naming.rebind(EVENT_MANAGER_NAME, this);
151      catch (Exception e) {
152          System.out.println(EventManagerImpl.can't bind
153              e.printStackTrace());
154          throw e;
155      }
156
157
158      /**
159       * Finalizer
160       */
161      /**
162       * Removes the EventManager's entry from the RMI Registry
163       */
164      protected void finalize()
165          throws Throwable {
166          System.out.println("EventManagerImpl Finalizer called");
167          /**
168           * Remove this EventManager's Name from the RMI Registry
169           */
170          Naming.unbind(EVENT_MANAGER_NAME);
171          catch (Exception e) {
172              System.out.println(EventManagerImpl.can't unbind
173                  e.printStackTrace());
174          }
175          /**
176           * Delete reference to Persistent Store Manager
177           */
178          PersistenceManager = null;
179          super.finalize();
180
181
182
183      /**
184       * Instance methods
185
186      /**
187       * Methods to add event registrations.
188      /**
189       * Register interest in receiving specified events.

```

- 224 -

EventManagerImpl.java\_1 Mon Nov 11 16:26:42 1996 5

```

189     * To remove these registrations use
190     * unregisterEvents(EventType events[]), EventPost eventQueue).
191     * @param events An array of the names of events that an object
192     * is interested in receiving.
193     * @param eventQueue the associated event queue's EventPost interface.
194   */
195   public synchronized void registerEvents(String events[], EventPost eventQueue)
196     throws IOException {
197
198     /*
199      * If the event queue has already registered to receive all
200      * notifications, ignore this request.
201      */
202     if (fullRegistrations.contains(eventQueue))
203       return;
204
205     /*
206      * Set the list of events the queue has registered
207      * for notifications. If the list does not exist, create it.
208      */
209     Vector notifyList = (Vector)eventRegistrations.get(eventQueue);
210
211     if (notifyList == null) {
212       notifyList = new Vector();
213     }
214
215     for (int i = 0; i < events.length; i++) {
216       String eventName = events[i];
217       System.out.println("registerEvents:Events, " + eventName);
218
219       /*
220        * Add the event to the queue's list of registered events.
221        */
222       if (!notifyList.contains(eventName)) {
223         System.out.println("registerEvents, adding event name to notify list");
224         notifyList.addElement(eventName);
225       }
226
227     /*
228      * Look up the event name in the eventRegistrations hash table
229      * and add the queue to the list of registered objects for the event.
230      */
231     Vector registryList = (Vector)eventRegistrations.get(eventName);
232
233     if (registryList == null) {
234       registryList = new Vector();
235     }
236     eventRegistrations.put(eventName, registryList);
237
238     System.out.println("registerEvents, adding event queue to registry list");
239

```

-225-

```

EventManagerImpl.java_1      Mon Nov 11 16:26:42 1996      6

        registryList.addElement(eventQueue);

    236
    237     ) else if (!registryList.contains(eventQueue)) {
    238         System.out.println("registerEvents, adding event queue to registry list");
    239         registryList.addElement(eventQueue);
    240     }
    241 }
    242 /*
    243     * Write the updated eventRegistrations table to
    244     * persistent storage.
    245     */
    246     updateEventRegistrations();
    247 }

    248
    249
    250 /**
    251     * Register interest in receiving all events posted to this EventManager.
    252     * To remove these registrations use
    253     * unregisterAll(EventPost eventQueue).
    254     * @param eventQueue The associated event queue's EventPost interface.
    255     */
    256 public synchronized void registerAll(EventPost eventQueue)
    257 throws IOException {
    258     System.out.println("registerAll");
    259     System.out.println("registerAll");
    260 }
    261 /*
    262     * If this event queue already registered to receive
    263     * specific registrations, reacive them.
    264     */
    265 if ((Vector)eventRegistrations.get(eventQueue) != null)
    266     unregisterAll(eventQueue);
    267
    268 /*
    269     * If the event queue has already been registered for
    270     * all notifications, ignore this request.
    271     * Otherwise add the the list of full registrations.
    272     */
    273 if (!fullRegistrations.contains(eventQueue))
    274     System.out.println("registerAll, adding event queue to full registrations");
    275     fullRegistrations.addElement(eventQueue);
    276 /*
    277     * Write the updated fullRegistrations list to
    278     * persistent storage.
    279     */
    280     updateFullRegistrations();
    281
    282 }

```

283     }

284  
285     // Methods to remove registrations.

286  
287     /\*\*

288         \* Remove registration for specified events.

289         \* @param events An array of the names of events that an object

290         \* is no longer interested in receiving.

291         \* @param eventQueue The associated event queue's EventPost interface.

292         \* @exception EventManagerException If the event queue was not registered

293         \* for one or more of the specified exceptions.

294         \*/

295     public synchronized void unregisterEvents(String events[], EventPost eventQueue)

296         throws EventManagerException, IOException {

297  
298         /\*

299             \* Get the list of events the queue has registered

300             \* for notifications.

301             \*/

302         Vector notifyList = (Vector)eventRegistrations.get(eventQueue);

303         if (notifyList == null)

304             throw new EventManagerException("Bad event queue");

305  
306         for (int i = 0; i < events.length; i++) {

307             String eventName = events[i];

308             System.out.println("unregisterEvents, event = " + eventName);

309  
310             /\*

311             \* Remove the event from the queue's list of registered events.

312             \*/

313             System.out.println("unregisterEvents, removing event name from notify list");

314             notifyList.removeElement(eventName);

315  
316             /\*

317             \* Lock up the event name in the eventRegistrations hash table

318             \* and remove the queue from the list of registered Objects for

319             \* the event. If the list is empty, remove the event's hash

320             \* table entry.

321             \*/

322         Vector registryList = (Vector)eventRegistrations.get(eventName);

323         if (registryList == null)

324             throw new EventManagerException("Bad event");

325  
326         System.out.println("unregisterEvents, removing event queue from registry list");

327         registryList.removeElement(eventQueue);

328  
329

- 227 -

## EventManagerImpl.java\_1      Mon Nov 11 16:26:42 1996      8

```

330     if (registryList.isEmpty()) {
331         System.out.println("unregisterEvents", removing hash table entry for event " + eventName);
332         eventRegistrations.remove(eventName);
333     }
334 }

335 /**
336 * Write the registered eventRegistrations table to
337 * persistent storage
338 */
339
340 updateEventRegistrations();

341 /**
342 * If the queue's list of registered events is empty, remove
343 * the queue's hash table entry.
344 */
345 if (registryList.isEmpty()) {
346     System.out.println("unregisterEvents", removing event queue's hash table entry");
347     eventRegistrations.remove(eventQueue);
348 }
349 }

350

351 /**
352 * Remove registration for all events
353 * @param eventQueue The associated event queue's EventPost interface.
354 * @param registrations The registrations held by finalizers, so it can't error.
355 * This method is often called by finalizers, so it can't error.
356 */
357 public synchronized void unregisterAll(EventPost eventQueue) {
358     Thread.interrupted();
359 }

360 if (eventRegistrations.contains(eventQueue)) {
361
362     /**
363      * If the event queue was registered to receive all registrations,
364      * remove it from the fullRegistrations hash table.
365      */
366     System.out.println("unregisterAll, removing full registration");
367     fullRegistrations.removeElement(eventQueue);
368
369     /**
370      * Write the updated fullRegistrations list to
371      * persistent storage.
372      */
373     updateFullRegistrations();
374 }
375
376     /**
377      * Locate the event queue's list of registered events.
378      */
379 }

```

EventManagerImpl.java\_1 Mon Nov 11 16:26:42 1996 9

```

377     */
378     Vector notifyList = (Vector)eventNotifications.get(eventQueue);
379     if (notifyList == null)
380       return;
381
382     System.out.println("unregisterAll, removing event registrations");
383
384     for (Enumeration e = notifyList.elements(); e.hasMoreElements(); ) {
385       String eventName = (String)e.nextElement();
386       System.out.println("unregisterAll, event = " + eventName);
387
388       Vector registryList = (Vector)eventRegistrations.get(eventName);
389       if (registryList == null)
390         continue;
391
392       /* For each event in the list, remove the queue from the
393        * event's list of registered Objects.
394       */
395       System.out.println("removing event queue from registry list");
396       registryList.removeElement(eventQueue);
397       if (registryList.isEmpty())
398         System.out.println("unregisterAll, removing hash table entry for event " + eventName);
399       eventRegistrations.remove(eventName);
400
401       registryList = null;
402
403
404
405       /*
406        * Verify the updated eventRegistrations table to
407        * guarantee storage.
408       */
409       updateEventRegistrations();
410
411       /*
412        * Remove the event queue's hash table entry.
413       */
414       System.out.println("unregisterAll, removing event queue's hash table entry");
415       eventNotifications.remove(eventQueue);
416
417     }
418
419     /*
420      * Methods to post events and send notifications
421     */
422     /**
423      * Notify objects that an event occurred (EventPost interface).

```

```

424     * @param event Event being posted.
425     * @exception EventException if an error occurs posting the event.
426   */
427   public synchronized void postEvent(EventType event)
428     throws PostException, EventException, IOException {
429     System.out.println("EventManager: Posting event");
430   }
431   /**
432    * Handle specific registrations for this event locate this event's
433    * entry in the eventRegistrations hash table and post the event to
434    * each of the event listeners in its hash chain.
435   */
436   String eventName = event.getClass().getName();
437   System.out.println("postEvent, event = " + eventName);
438   System.out.println("Registration, checking specific registrations");
439
440   /**
441    * Post one event to the event queues registered to
442    * receive it.
443   */
444   Vector listenerRegistrations = new Vector();
445
446   Vector registryList = (Vector)eventRegistrations.get(eventName);
447   if (registryList == null) {
448     System.out.println("registryList contains " + registryList.size());
449   }
450
451   for (Enumeration e = registryList.elements(); e.hasMoreElements(); ) {
452     try {
453       EventPostable epe = (EventPostable)e.nextElement();
454       System.out.println("posting event");
455
456       epe.postEvent(event);
457     } catch (Exception e2) {
458       System.out.println("Event Post failed");
459       epe.removeListener(registrations.addElement(eventQueue));
460     }
461   }
462
463
464   /**
465    * Post the event to all objects that have requested notification
466    * of all events.
467   */
468   System.out.println("postEvent, checking full registrations");
469
470 
```

- 230 -

EventManagerImpl.java\_1 Mon Nov 11 16:26:42 1996 11

```

471     System.out.println("fullRegistrations contains " + fullRegistrations.size() +
472         " elements");
473
474     Vector badFullRegistrations = new Vector();
475     for (Enumeration e = fullRegistrations.elements(); e.hasMoreElements(); ) {
476         EventPost eventQueue = (EventPost)e.nextElement();
477
478         System.out.println("Posting event");
479         try {
480             eventQueue.postEvent(event);
481         } catch (Exception e3) {
482             System.out.println("Event Post failed");
483             badFullRegistrations.addElement(eventQueue);
484         }
485
486         /*
487          * if the EventManager was unable to contact any event queues,
488          * remove them from eventRegistrations and/or fullRegistrations
489          * and update the affected registrations in persistent storage.
490        */
491
492         if (!badEventRegistrations.isEmpty()) {
493             for (Enumeration e = badEventRegistrations.elements(); e.hasMoreElements(); ) {
494                 EventPost eventQueue = (EventPost)e.nextElement();
495                 System.out.println("Removing bad event queue from registry list");
496                 registrationList.removeElement(eventQueue);
497
498                 updateEventRegistrations();
499             }
500         }
501
502         if (badFullRegistrations.isEmpty()) {
503             for (Enumeration e = badFullRegistrations.elements(); e.hasMoreElements(); ) {
504                 EventPost eventQueue = (EventPost)e.nextElement();
505                 System.out.println("Removing bad event queue from full list");
506                 registrationList.removeElement(eventQueue);
507
508                 updateFullRegistrations();
509             }
510         }
511     }
512
513
514     // Methods to handle persistence.
515
516
517     /**

```

```

518     * Allocate a persistent store manager and restore
519     * eventRegistrations and fullRegistrations. (If they
520     * don't exist in the persistent store, instantiate
521     * new ones.) Then reconstruct eventNotifications from
522     * eventRegistrations.
523
524     protected void reconstructRegistrations() {
525
526         Boolean enabled = new Boolean(getProperty(ENABLE_PERSISTENCE, "false"));
527
528         /*
529          * If persistence is disabled, perform initialization,
530          * but not necessary.
531          */
532         System.out.println("recoverRegistrations: persistence disabled");
533         persistenceStorefile = "";
534         persistenceManager = null;
535         eventRegistrations = new Hashtable();
536         fullRegistrations = new Vector();
537         eventNotifications = new Hashtable();
538
539         System.out.println("persistence enabled.");
540         PersistenceStorefile = getProperty(PERSISTENCE_FILE, "");
541         System.out.println("EventManager.persistence.storefile property set to: " + persistenceStorefile);
542
543         /*
544          * If no file is specified, use the default
545          */
546         if (persistenceStorefile.length() == 0)
547             persistenceStorefile = AgentConfig.Agent_DIR + DEFAULT_PERSISTENCE_FILE;
548
549         System.out.println("Persistent Store File: " + persistenceStorefile);
550
551         persistenceManager = new PersistentStoreHandler(persistenceStorefile);
552
553         /*
554          * Fetch registrations from the persistent store. If they don't exist,
555          * instantiate eventRegistrations and fullRegistrations.
556          */
557         System.out.println("Fetching eventRegistrations");
558         if (eventRegistrations == null)
559             eventRegistrations = new HashTable();
560         eventRegistrations = new Hashtable();
561
562         System.out.println("Fetching fullRegistrations");
563         if (fullRegistrations == null)
564             fullRegistrations = new Vector();

```

EventManagerImpl.java\_1 Mon Nov 21 16:26:42 1996 13

```

565     (Vector)persistenceManager.fetchObject(FULL_REGISTRATION_ID)) == null)
566     fullRegistrations = new Vector();
567
568     /*
569      * Reconstruct the event notifications using event registrations. Then
570      * regenerate the persistent store and write out the registrations.
571      */
572     System.out.println("Reconstructing event notifications");
573     reconstructEventRegistrations();
574
575     System.out.println("Saving registrations");
576     saveRegistrations();
577 }
578
579 /**
580  * Truncate persistent storage and write out event and full
581  * registrations
582  */
583
584 protected void saveRegistrations() {
585     if (persistenceManager == null)
586         return;
587
588     persistenceManager.truncateFilestore();
589     persistenceManager.insertObject(EVENT_REGISTRATION_ID,
590     eventRegistrations, STORAGE_ADDING);
591     persistenceManager.insertObject(FULL_REGISTRATION_ID,
592     fullRegistrations, STORAGE_PERSISTING);
593
594
595 /**
596  * Update event registrations in persistent storage.
597  */
598
599 protected void updateEventRegistrations() {
600     if (persistenceManager == null)
601         return;
602
603     System.out.println("Updating eventRegistrations");
604     persistenceManager.updateObject(EVENT_REGISTRATION_ID, eventRegistrations);
605
606
607 /**
608  * Update full registrations in persistent storage.
609  */
610
611 protected void updateFullRegistrations() {

```

```

612     if (persistenceManager == null)
613         return;
614
615     System.out.println("Updating fullRegistrations");
616     persistenceManager.updateObject(FULL_REGISTRATION_ID, fullRegistrations);
617 }
618
619 /**
620  * Reconstruct the event notifications table from the
621  * event registrations table.
622  */
623 protected void reconstructNotifications() {
624     eventNotifications = new Hashtable();
625
626     if (persistenceManager == null)
627         return;
628
629 /**
630  * Reconstruct the event notifications hash table by walking
631  * the event registrations hash table and extracting registrations.
632  */
633
634 Enumeration registryKeys = eventRegistrations.keys();
635 Vector registryElements = eventRegistrations.elements();
636
637 /**
638  * For each key (event name) in the table, obtain the list of
639  * registered event names.
640  */
641 String eventName = (String)registryKeys.nextElement();
642 System.out.println("reconstructNotifications: event = " + eventName);
643 Vector registryList = (Vector)eventRegistrations.get(eventName);
644
645 /**
646  * If no objects have registered to receive this event,
647  * remove it from the eventRegistrations hash table.
648  */
649 if (registryList == null) || (registryList.isEmpty()) {
650     System.out.println("reconstructNotifications: removing hash table entry for event " + eventName);
651     eventRegistrations.remove(eventName);
652     continue;
653 }
654
655 /**
656  * Add the event name to the eventRegistrations entry of
657  * each of the registered event queues, creating hash table
658  * entries, as necessary.

```

```

559      */
560      for (Enumeration registryElements = registryList.elements();
561          registryElements.hasMoreElements(); ) {
562
563          EventPost eventQueue = (EventPost) registryElements.nextElement();
564          Vector notifyList = (Vector) eventQueue.notifications.get(eventQueue);
565
566          if (notifyList == null) {
567              notifyList = new Vector();
568              eventQueue.notifications.put(eventQueue, notifyList);
569              System.out.println("reconstructNotifications, created eventNotifications hash table entry");
570          }
571
572      }
573
574      /*
575      /**
576      /** Returns the EventManager's name */
577      protected static String getEventManagerName() {
578          return EVENT_MANAGER_NAME;
579      }
580
581      /**
582      /** Returns the value of a property. */
583      protected String getProperty(String key, String def) {
584          return preferences.getProperty(key, def);
585      }
586
587      /**
588      /**
589      * EventManagerImpl constructor. This is the only way to start
590      * the EventManager since it has no public constructors.
591      */
592      public static void main(String args[]) {
593          System.out.println("EventManagerImpl(" + args[0] + ")");
594
595          EventManagerImpl eventManager = new EventManagerImpl();
596
597          try {
598              catch (Exception e, {
599                  System.out.println("An exception occurred setting up the EventManager: " +
600                      e.getMessage());
601
602                  e.printStackTrace();
603
604              }
605          }

```

```

EventManagerProxy.java_1           Mon Nov 11 16:26:42 1996      1   10/22/96 7:30p Noemi $ 
1  /* SHeader: /com/meitca/hsl/zcnesagents/event/EventManagerProxy.java 1
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC IIA.
8  */
9  * DESCRIPTION
10 * A client proxy for the EventManager.
11 * 
12 * $Log:EventManager.java,v $ 1.1 10/22/96 7:25p Noemi
13 * 
14 * EventManagerProxy is the public interface to the EventManager. It
15 * shields users from the effects of EventManager crashes.
16 * 
17 */
18 package com.meitca.hsl.zcnesagents.event;
19 
20 import java.io.*;
21 import java.rmi.*;
22 import java.rmi.RemoteException;
23 import java.rmi.Naming;
24 import java.rmi.Remote;
25 
26 /**
27 * EventManagerProxy is the public interface to the EventManager as a
28 * proxy for the Event Manager. It maintains a reference to the actual
29 * Event Manager and shields users from the effects of Event Manager
30 * crashes. Whenever the proxy fails to communicate with the Event Manager
31 * it attempts to re-establish a connection to it.
32 * 
33 * @see EventManager
34 * @see EventManagerImpl
35 * @author Noemi Pachter
36 * 
37 */
38 
39 public class EventManagerProxy {
40 
41     // Constants
42     protected static final int DEFAULT_RETRYES = 2;
43     protected static final long DEFAULT_TIMEOUT = 10000;
44 
45     // Instance variables
46 
```

```

EventManagerProxy.java_1    Mon Nov 11 16:26:42 1996   2
48  /** Stub for EventManagerImpl's EventManager interface. */
49  protected EventManager eventManager;
50
51  /** Stub for EventManagerImpl's EventPost interface. */
52  protected EventPost eventPost;
53
54  /** The EventManagerImpl's name. */
55  protected String eventManagerName;
56
57  /**
58   * The number of times to retry when an invocation
59   * on the EventManager fails.
60   */
61  int retries;
62
63  /** The timeout (in ms) for requests to the EventManager. */
64  long timeout;
65
66  /**
67   * Constructor
68   * @param EventManagerProxy
69   * @throws IOException
70   */
71  public EventManagerProxy()
72  throws IOException {
73
74      this(DEFAULT_RETRIES, DEFAULT_TIMEOUT);
75
76
77
78  /**
79   * Constructs an EventManagerProxy
80   * @param timeout the number of times to retry if an operation fails.
81   */
82  public EventManagerProxy(int retries)
83  throws IOException {
84
85      this(retries, DEFAULT_TIMEOUT);
86
87
88
89  /**
90   * Constructs an EventManagerProxy
91   * @param timeout the time to wait in ms. between retries.
92   */
93  public EventManagerProxy(long timeout)
94  throws IOException {

```

```

95     this(DEFAULT_RETRIES, timeout);
96
97 }
98
99 /**
100  * Constructs an EventManagerProxy.
101  * @param retries The number of times to retry if an operation fails.
102  * @param timeout The time to wait in ms. between retries.
103  */
104 public EventManagerProxy(int retries, long timeout)
105 throws Exception {
106
107     this.retries = (retries >= 0) ? retries : DEFAULT_ATTEMPTS;
108     this.timeout = (timeout > 0) ? timeout : DEFAULT_TIMEOUT;
109
110
111     /*
112      * Lookup the EventManager.
113      */
114     initialize();
115
116
117     /*
118     * Instance field diag
119     */
120     /*
121      * Lookup the Event Manager in the RM2 registry and obtain a
122      * stub for the EventManager and EventPost interfaces.
123      */
124     private void initialize()
125     throws Exception {
126
127         eventManagerName = EventManagerImpl.getEventManagerName();
128
129         RemoteStub = Naming.lookup(EventManagerName);
130
131         eventManager = (EventManager) stub;
132         eventPost = (EventPost) stub;
133
134         catch (Exception e) {
135             System.out.println("initialize error looking up EventManager: "
136                         + e.getMessage());
137             e.printStackTrace();
138             throw e;
139         }
140
141         /*
142          * Block for the timeout interval. Then look up the

```

```

4
EventManagerProxy.java_1      Mon Nov 11 16:26:42 1996

142     * EventManager.
143     */
144     private void waitAndInitialize()
145         throws Exception {
146
147     synchronized(this) {
148         try {
149             this.wait(timeout);
150         } catch (InterruptedException e) {
151         }
152     }
153 }
154
155     /*
156     * Convert the event class names to strings.
157     */
158     private String[] eventNamesToString(EventType[] events) {
159
160         int length = events.length;
161         String eventNames[] = new String[length];
162         for (int i = 0; i < length; i++) {
163             eventNames[i] = events[i].getClassName();
164             System.out.println("eventName: " + eventNames[i]);
165         }
166     return eventNames;
167 }
168
169
170     /**
171      * Method to handle registrations.
172      */
173     /*
174     * Register interest in receiving specified events.
175     * param events An array of events an object is interested in receiving.
176     * param queue The associated event queue's EventPost interface.
177     * exception RuntimeException if an error occurs setting up network
178     * connections.
179     * exception IOException if the transport layer generates one.
180     */
181     public void registerEvents(EventType events[], EventPost eventQueue)
182         throws Exception {
183
184     /*
185     * Convert the event class names to strings.
186     * Then invoke the EventManager's registerEvents
187     * method and retry the operation, if necessary.
188     */

```

- 239 -

```

EventManagerProxy.java_1    Non Nov 11 16:26:42 1996   5

189     String eventNames[] = eventNamesToString(events);
190     for (int i = 0; i < retries; i++) {
191         try {
192             eventManager.registerEvents(eventNames, eventQueue);
193             return;
194         } catch (RemoteException e1) {
195             if (i == retries) {
196                 System.out.println("registerEvents: Throwing RemoteException");
197                 throw e1;
198             }
199             System.out.println("registerEvents: Caught RemoteException");
200             catch (IOException e2) {
201                 if (i == retries) {
202                     System.out.println("registerEvents: Throwing IOException");
203                     throw e2;
204                 }
205                 System.out.println("registerEvents: Caught IOException");
206             }
207             /*
208             * Block for one timeout interval. Then look up the
209             * EventManager, before retrying the operation.
210             */
211             waitInitialization();
212         }
213     }
214
215
216     /*
217     * Register interest in receiving all events posted to this EventManager.
218     * Generate eventQueue. The associated event queue's EventPost interface.
219     * RemoteException: Retries exception if an error occurs setting up network
220     * connections.
221     * IOException: IOException if the transport layer generates one.
222     */
223     public void registerAll(EventPost eventQueue)
224     throws Exception {
225
226     /*
227     * Invoke the EventManager's registerAll
228     * method and retry the operation, if necessary.
229     */
230     for (int i = 0; i < retries; i++) {
231         try {
232             eventManager.registerAll(eventQueue);
233             return;
234         } catch (RemoteException e1) {
235             if (i == retries) {

```

-240-

```

EventManagerProxy.java_1      Mon Nov 11 16:26:42 1996   6
System.out.println("registerAll: Throwing RemoteException");
    }

    System.out.println("registerAll: Caught RemoteException");
    }

    catch (IOException e2) {
        if (i < retries) {
            System.out.println("registerAll: Throwing IOException");
            throw e2;
        }
        System.out.println("registerAll: Caught IOException");
    }

    /*
     * Block for one timeout interval. Then look up the
     * EventMaster, before retrying the operation.
     */
    waitAndInitiate();
}

}

/*
Methods to deregister events.
*/
/*
 * Remove registration for specified events.
 * An array of events an object is interested in receiving.
 * param events
 * param EventPost interface.
 */
public void deregisterEvents(EventNames[] events)
throws RemoteException {
    /*
     * Convert the event class names to strings
     * Then invoke the EventManager's registerEvents
     * method and return one operation, if necessary.
     */
    String eventNames[] = eventNameToString(events);
    for (int i = 0; i < retries; i++) {
        try {
            eventManager.deregisterEvents(eventNames, eventQueue);
        return;
        } catch (RemoteException e1) {
            if (i == retries) {
                System.out.println("registerEvents: Throwing RemoteException");
}

```

```

283     throw e1;
284
285     System.out.println("unregisterEvents: Caught RemoteException");
286   } catch (IOException e2) {
287     if (i < retries) {
288       System.out.println("unregisterEvents: Throwing IOException");
289       throw e2;
290     }
291     System.out.println("unregisterEvents: Caught IOException");
292
293   /* Block for the timeout interval. Then look up the
294    * EventManager. Before retrying the operation.
295    */
296   waitAndInitialize();
297
298 }
299
300 /**
301  * Remove registration for all events.
302  * @param eventQueue The associated event queue's EventPost interface.
303  * This method is often called by finalizers, so it does not
304  * propagate any errors.
305  */
306 public void unregisterAll(EventPost eventQueue) {
307
308   /* Invoke the EventManager's registerAll
309   * method and retry the operation, if necessary.
310   */
311   try {
312     for (int i = retries; i-- > -1) {
313       eventManager.unregisterAll(eventQueue);
314     }
315   } catch (RemoteException e1) {
316     System.out.println("unregisterAll: Caught RemoteException");
317   } catch (IOException e2) {
318     System.out.println("unregisterAll: Caught IOException");
319   }
320
321   /* Block for the timeout interval. Then look up the
322    * EventManager, before retrying the operation.
323    */
324   try {
325     waitAndInitialize();
326   } catch (Exception e3) {
327     System.out.println("waitAndInitialize failed");
328   }
329 }
```

- 242 -

```

EventManagerProxy.java_1      Mon Nov 11 16:26:42 1996      8

    }
}

330
331
332
333
334 // Methods to post events.
335
336 /**
337 * Event notification interface.
338 * @param event The event to post.
339 * @exception RemoteException If an error occurs setting up network
340 * connections.
341 * @exception EventException If an error occurs posting the event.
342 * @exception IOException If an exception in the transport layer generates one.
343 */
344 public void postEvent(EventType event)
345 throws Exception {
346 /**
347 * Invoke the EventManager's eventPost
348 * method and retry the operation, if necessary.
349 * method.
350 * @param i retries; i--.
351 */
352 try {
353     eventPost.postEvent(event);
354     return;
355 } catch (RemoteException e1) {
356     if (i == retries) {
357         System.out.println("postEvent: Throwing RemoteException");
358         throw e1;
359     }
360     System.out.println("postEvent: Caught RemoteException");
361     catch (IOException e2) {
362         if (i == retries) {
363             System.out.println("postEvent: Throwing IOException");
364             throw e2;
365         }
366         System.out.println("postEvent: Caught IOException");
367     }
368 }
369 /**
370 * Block for the time-out interval. Then look up the
371 * EventManager, before retrying the operation.
372 */
373 waitAndInitialize();
374
375 }
376

```

- 243 -

9

EventManagerProxy.java\_1      Mon Nov 11 16:26:42 1996

EventManagerProxy.java\_1

377

- 245 -

```

EventPost.java_1      Mon Nov 11 16:26:42 1996      1

1  /*
2   *  SHheader: /com/meitcar/bellmesagents/event/EventPost.java 2
3   *  Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4   *  Information Technology Center America.
5   *  All rights reserved.
6   *
7   *  CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8   *
9   *  DESCRIPTION
10  *  Interface for event notification
11  *
12  *  $Log: /com/meitcar/bellmesagents/event/EventPost.java 5
13  *  2 10/22/96 7:10p Noemi
14  *  Changed the third clause of the postEvent method.
15  *
16  *  1 10/14/96 6:51p Noemi
17  *  Interface used for getting agents.
18  *
19  *
20 package com.meitcar.bellmesagents.event;
21
22 import java.io.*;
23
24 import java.rmi.Remote;
25 import java.rmi.RemoteException;
26
27 /**
28  * EventPost is a Remote interface for notification of events.
29  * Author: Noemi Pachon
30  *
31  */
32
33 public interface EventPost
34 extends Remote {
35
36 /**
37  * instance methods.
38  */
39 /**
40  * Event notification interface.
41  * param event The event to post.
42  * gexception NetworkException if an error occurs setting up network
43  * connections.
44  * gexception EventException If an error occurs posting the event.
45  */
46 public void postEvent(EventType event)
47 throws RemoteException, EventException, IOException;

```

2

EventPost.java\_1 Mon Nov 11 15:25:42 1996

EventPost.java\_1

48 }

EventQueueImpl.java\_1      Mon Nov 11 16:26:42 1996      1

1    /\* \$Header: /com/mitsc/ca/hsl/zonesagents/event/EventQueueImpl.java 1      10/14/96 6:58p Noemi \$

2    \*      DESCRIPTION      Event queue class definition.

3    \*      Copyright 1995 Horizon Systems Laboratory, Mitsubishi Electric

4    \*      Information Technology Center America.

5    \*      All rights reserved.

6    \*      CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA

7    \*/

8    package com.mitsc.ca.hsl.zonesagents.event;

9    import java.util.\*;

10   import java.rmi.server.UnicastRemoteServer;

11   import java.rmi.RemoteException;

12   import java.rmi.Naming;

13   import java.rmi.NotBoundException;

14   import java.util.Date;

15   import java.util.Vector;

16   import java.util.List;

17   import com.mitsc.ca.hsl.zonesagents.event;

18   import java.util.List;

19   import java.util.ArrayList;

20   import java.util.ListIterator;

21   import java.util.ListIterator;

22   import java.rmi.server.UnicastRemoteServer;

23   import java.rmi.RemoteException;

24   import java.rmi.Naming;

25   import java.rmi.NotBoundException;

26   /\*\*

27   \* EventQueueImpl is an event queue that implements the RMI

28   \* EventPost interface.

29   \*

30   \* Event notification is performed by appending an event to the queue.

31   \* Each queue has an associated EventQueueThread started by the constructor.

32   \* The thread dequeues events and calls their handlers.

33   \*

34   \* @see EventQueueThread

35   \* @see EventPost

36   \* @author Noemi Faricrek

37   \*

38   \*/

39   public class EventQueueImpl

40   extends UnicastRemoteServer

41   implements EventPost {

42   }

43   // instance variables

44   /\*\* The event queue \*/

45   protected Vector eventQueue;

46   }

47   }

- 267 -

```

EventQueueImpl.java_1   Mon Nov 11 16:26:42 1996    2

48  /** The thread that manages this queue */
49  protected EventQueueThread queueThread;
50
51  /**
52   * Protected Default Constructor for EventQueueImpl.
53   * @exception IllegalStateException whenever called.
54   */
55  protected EventQueueImpl()
56  throws RemoteException, IllegalStateException {
57  System.out.println("Default Constructor called for EventQueueImpl");
58  throw new IllegalStateException("default constructor");
59 }
60
61 /**
62  * Constructs an EventQueueImpl.
63  * @param handler An EventHandler interface.
64  */
65 public EventQueueImpl(EventHandler handler)
66 throws RemoteException {
67 eventQueue = new Vector();
68 queueThread = new EventQueueThread(this, handler);
69 }
70
71 /**
72  * Constructs an EventQueueImpl.
73  * @param handler An EventHandler interface.
74  */
75 public EventQueueImpl(EventHandler handler, long timeout)
76 throws RemoteException {
77 eventQueue = new Vector();
78 queueThread = new EventQueueThread(this, handler, timeout);
79 }
80
81 /**
82  */
83
84 /**
85  */
86 /**
87  * Posts an event by enqueueing it and waking up the
88  * queue's thread.
89  * @param event The event to post.
90  */
91 public synchronized void postEvent(EventType event) {
92  System.out.println("EventQueueImpl posting event");
93  enqueue(event);
94  this.notifyAll();
}

```

```

EventQueueImpl.java_1      Mon Nov 11 16:26:42 1996      3

95
96
97    /**
98     * Enqueues an event.
99     * @param event The event to queue.
100    */
101   protected final synchronized void enqueue(EventType event) {
102     eventQueue.addElement(event);
103   }
104
105   /**
106    * Dequeues an event.
107    * @return The first event in the queue.
108    */
109   protected final synchronized EventType dequeue() throws ArrayIndexOutOfBoundsException {
110     if (!eventQueue.isEmpty()) {
111       EventType event = (EventType)eventQueue.firstElement();
112       eventQueue.removeElementAt(0);
113       return event;
114     }
115     return null;
116   }
117
118
119
120
121
122
123
124
125
126
127

```

```

EventQueueThread.java_1   Mon Nov 11 16:26:42 1996   1
1  /* $Header: /com/meitca/hs1/zones/jagents/event/EventQueueThread.java 1      10/14/96 6:59p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  */
7  */
8  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA
9  */
10 */
11 */
12 */
13 */
14 */
15 */
16 */
17 */
18 package com.meitca.zones.jagents.event;
19 */
20 */
21 */
22 */
23 */
24 */
25 */
26 */
27 */
28 */
29 */
30 */
31 */
32 */
33 */
34 */
35 */
36 */
37 */
38 */
39 */
40 */
41 */
42 */
43 */
44 */
45 */
46 */
47 */

/*
 * EventQueueThread is a Thread that manages an event queue.
 * Event notification is implemented by appending an event to the queue
 * and waking up its EventQueueThread to process the event by calling its
 * event handler.
 * The EventQueueThread is instantiated by the EventQueueImpl constructor.
 *
 * @see EventQueueImpl
 * @see EventPost
 * @see EventHandler
 * @author Noemi Pacharek
 */
class EventQueueThread
    extends Thread {
}

// Constants
/** The default timeout value in ms. */
public static final long DEFAULT_TIMEOUT = 1000;

// Instance variables
/** This thread's event queue */
protected EventQueue eventQueue;
/** The application's event handler */
protected EventHandler eventHandler;

/** The time in ms between consecutive examinations of the queue. */
protected long timeout;

```

```

EventQueueThread.java_1      Mon Nov 1 16:26:42 1996      2

48     /**
49      * Constructors
50      */
51      * Protected Default Constructor for an EventQueueThread.
52      *      * Exception illegalAccess exception whenever called.
53      */
54      protected EventQueueThread()
55      throws IllegalAccessException {
56      System.out.println("Default constructor for EventQueueThread");
57      new IllegalAccessException("EventQueueThread default constructor");
58
59
60      /**
61       * Constructs an EventQueueThread.
62       *      * Queue The associated event queue.
63       *      * Handler the associated event handler interface.
64       */
65      public EventQueueThread(EventQueueImpl queue, EventHandler handler) {
66      this(queue, handler, DEFAULT_TIMEOUT);
67
68
69      /**
70       * Constructs an EventQueueThread.
71       *      * Queue The associated event queue.
72       *      * Handler the associated event handler interface.
73       *      * Timeout the time in ms to wait between consecutive queue
74       *      * examinations.
75       */
76      public EventQueueThread(EventQueueImpl queue, EventHandler handler, long timeout) {
77      eventQueue = queue;
78      eventHandler = handler;
79      this.timeout = timeout;
80      this.start();
81
82      /**
83       * Instance metric is
84      */
85      /**
86       * The thread's run method.
87       */
88      public void run() {
89      try {
90      synchronized(eventQueue) {
91      eventQueue.wait(timeout);
92      } catch (InterruptedException e) {
93      }
94

```

- 251 -

EventQueueThread.java\_1 Mon Nov 11 16:26:42 1996 3

```
95     EventType event;
96     while (event = eventQueue.dequeue() != null) {
97         System.out.println("EventQueueThread, handling event:");
98         try {
99             eventHandler.handleEvent(event);
100         } catch (Exception ex) {
101             System.out.println("EventQueueThread: can't handle event: " +
102                         ex.getMessage());
103             ex.printStackTrace();
104         }
105     }
106 }
107 }
108 }
109 }
110 }
```

- 252 -

```

EventType.java_1    Mon Nov 11 16:25:43 1996   1
1  /* $Header: /ccm/meitca/hs1/zmesagents/event/EventType.java 1      10/14/96 6:55p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8  */
9  * DESCRIPTION
10 * Base class for distributed events.
11 *
12 * $Log: /com/meitca/hs1/zmesagents/event/EventType.java $
13 *
14 * 1 10/14/96 6:55p Noemi
15 * Base class for distributed events.
16 */
17
18 package com.meitca.hsi.zmesagents.event;
19
20 import com.meitca.hsi.util.UnitedID;
21
22 /**
23 * Base class for distributed events. EventType describes an event.
24 * It contains: a unique ID (generated by the EventManager) and a description
25 * (optionally passed to the constructors)
26 *
27 * Applications must derive and handle their own event types.
28 *
29 * Author: Noemi Pariente
30 */
31
32 abstract public class EventType {
33
34     // Constants
35     private static final String UNKNOWN = "Unknown";
36     private static final String TYPE = "event";
37
38     // Instance variables
39     /** A unique ID representing the event */
40     protected String eventID;
41
42     /** A description of the event */
43     protected String description;
44
45     // Constructors
46     /** Constructs an EventType */
47

```

- 253 -

```

EventType.java_1      Mon Nov 11 16:26:43 1996      2

48     public EventType() {
49         this.UNKNOWN;
50     }
51
52     /** Constructs an EventType
53      * @param description A String describing the event.
54      */
55     public EventType(String description) {
56         eventId = UniqueID.GenerateID(TYPE, this);
57         System.out.println("eventId = " + eventId);
58         this.description = description;
59     }
60
61     // Instance methods
62
63     /**
64      * String representation of event.
65      * @return A String representation of the event.
66      */
67     public final String toString() {
68         return (eventId + " " + description);
69     }
70
71     /**
72      * Retrieve an event's ID.
73      * @return The event's ID.
74      */
75     public final String getEventID() {
76         return eventId;
77     }
78
79     /**
80      * Retrieve an event's description.
81      * @return A description of the event.
82      */
83     public final String getEventDescription() {
84         return description;
85     }
86
87     }
88
89
90
91
92
93
94

```

- 254 -

3

EventType.java\_1      Mon Nov 11 16:25:43 1996

95  
96  
97  
98

Mon Nov 11 16:23:18 1996 1

```
1  * $Header: /com/meitca/hs1/zonesagents/shared/PersistentStoreHandler.java 1 10/22/96 7:32p Noemi $
2  *
3  * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4  * Information Technology Center America.
5  * All rights reserved.
6  *
7  * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC 7A
8  *
9  * DESCRIPTION
10 * persistent Store Manager wrapper that handles exceptions.
11 * $Log: /com/meitca/hs1/zonesagents/shared/PersistentStoreHandler.java $
12 * Revision 1.10 10/22/96 7:32p Noemi
13 * A generic wrapper for PersistentStoreManager that catches exceptions
14 * and retries operations, as necessary.
15 *
16 */
17
18 package com.meitca.hs1.zonesagents.shared;
19
20 import com.meitca.hs1.zonesagents.persistence.*;
21
22 /**
23 * A wrapper class for the persistent storage manager
24 * (PersistentStoreManager) that catches
25 * all its exceptions and performs the appropriate cleanup.
26 * @see PersistentStoreManager
27 * @author Noemi Pascual
28 */
29
30 public class PersistentStoreHandler {
31
32 /**
33 * Instance variables
34 * A reference to the persistent store manager.
35 * protected PersistentStoreManager persistenceManager;
36
37 /**
38 * Constructs a PersistentStoreHandler
39 * @param filename The name of the persistent storage file.
40 */
41
42 public PersistentStoreHandler(String filename) {
43     try {
44         persistenceManager = new PersistentStoreManager(filename);
45     } catch (PersistentStoreException e) {
46         System.out.println("PersistentStoreHandler: can't create persistent store manager " + e.getMessage());
47         persistenceManager = null;
48     }
49 }
```

- 256 -

```

48
49
50    }
51    // Finalizer
52    /**
53     * Deletes reference to PersistentStoreManager.
54     */
55    protected void finalize()
56        throws Throwable {
57        System.out.println("PersistentStoreHandler Finalizer called");
58
59        // Delete reference to Persistent Store Manager.
60        persistentManager = null;
61
62        super.finalize();
63
64    }
65
66
67    /**
68     * Instance methods.
69     */
70    /**
71     * Find an object in the persist store.
72     * @param object The Object to locate.
73     * @return The Object's persistent storage ID.
74     */
75    public int findObject(Object object) {
76        return persistentManager.findObject(object);
77    }
78
79    /**
80     * Insert an Object into persistent storage.
81     * If a storage manager exception occurs, attempt to
82     * truncate the persistent storage and disable persistence.
83     * @param object The Object represented by id.
84     * @return The Object's persistent storage ID.
85     * A return value of PersistentStoreManager.INVALID_OID indicates
86     * an unsuccessful attempt to insert the object.
87
88    public int insertObject(Object object) {
89        return insertObject(0, object, 0);
90    }
91
92    /**
93     * Insert an Object into persistent storage.
94     * If a storage manager exception occurs, attempt to

```

- 247 -

Mon Nov 11 16:23:18 1996 3

```

95      * Truncate the persistent storage and disable persistence.
96      * @param Object The object represented by id.
97      * @param space The additional space in bytes to be preallocated
98      * for this object.
99      * @return The object's persistent storage ID.
100     * A return value of PersistentStoreManager.INVALID_CID indicates
101     * an unsuccessful attempt to insert the object.
102
103    public int insertObject(Object object, int space) {
104        return insertObject(0, object, space);
105    }
106
107
108    /**
109     * Insert an object into persistent storage.
110     * If a storage manager exception occurs, attempt to
111     * truncate the persistent storage and disable persistence.
112     * @param id The object's persistent storage ID.
113     * @param object The object represented by id.
114     * @return The object's persistent storage ID.
115     * A return value of PersistentStoreManager.INVALID_CID indicates
116     * an unsuccessful attempt to insert the object.
117     */
118    public int insertObject(int id, Object object) {
119        return insertObject(id, object, 0);
120    }
121
122
123    /**
124     * Insert an object into persistent storage.
125     * If a storage manager exception occurs, attempt to
126     * truncate the persistent storage and disable persistence.
127     * @param id The object's persistent storage ID.
128     * @param object The object represented by id.
129     * @param space The additional space in bytes to be preallocated
130     * for this object.
131     * A return value of PersistentStoreManager.INVALID_OID indicates
132     * an unsuccessful attempt to insert the object.
133     */
134    public int insertObject(int id, Object object, int space) {
135        if (persistenceManager == null)
136            return PersistenceStoreManager.INVALID_OID;
137
138        try {
139            if (id == 0)
140                return persistenceManager.insertObject(object, space);
141            return persistenceManager.insertObject(id, object, space);

```

```

142     } catch (PersistentStoreDuplicateObjectException e1) {
143         try {
144             id = persistenceManager.findObject(object);
145             if (id != PersistentStoreManager.INVALID_OID) {
146                 persistenceManager.updateObject(id, object);
147                 return id;
148             }
149             catch (PersistentStoreException e2) {
150                 System.out.println("insertObject: duplicate object found" + e2.getMessage());
151             e2.printStackTrace();
152             truncateFileStore();
153             persistenceManager = null;
154         }
155         catch (PersistentStoreInvalidObjectException e3) {
156             System.out.println("insertObject: invalid ID" + e3.getMessage());
157         }
158         catch (PersistentStoreException e4) {
159             System.out.println("insertObject: can't insert object " + e4.getMessage());
160             e4.printStackTrace();
161             truncateFileStore();
162             persistenceManager = null;
163         }
164         return PersistentStoreManager.INVALID_OID;
165     }
166
167     /**
168      * Fetch an object from persistent storage.
169      * If a storage manager exception occurs, attempt to
170      * truncate the persistent storage and disable persistence.
171      * @param id the object's persistent storage ID.
172      * @return Object. The object represented by id, or null,
173      * if the object was not found in persistent storage.
174      */
175     public Object fetchObject(int id) {
176         if (persistenceManager == null)
177             return null;
178
179         try {
180             return persistenceManager.fetchObject(id);
181         }
182         catch (PersistentStoreObjectNotFoundException e1) {
183             System.out.println("fetchObject: Object not found in persistent store, ID = " + id);
184         }
185         catch (PersistentStoreInvalidObjectException e2) {
186             System.out.println("fetchObject: invalid ID" + e2.getMessage());
187         }
188         catch (PersistentStoreException e3) {
189             System.out.println("fetchObject: can't fetch object " + e3.getMessage());

```

- 255 -

```

Mon Nov 11 16:23:13 1996      5

189   e2.printStackTrace();
190   truncateFileStore();
191   persistenceManager = null;
192 }
193 return null;
194 }

195 /**
196  * Update an object in persistent storage.
197  * If a storage manager exception occurs, attempt to
198  * truncate the persistent storage and disable persistence.
199  */
200  * @param id          The Object's persistent storage ID.
201  * @param object      The updated Object represented by id.
202  * @return            A boolean indicating if the update was successful.
203
204 public boolean updateObject(int id, Object object) {
205     if (persistenceManager == null)
206         return false;
207
208     try {
209         persistenceManager.updateObject(id, object);
210     } catch (Exception e1) {
211         System.out.println("updateObject: Object not found in persistent store, ID = " + id);
212     }
213     try {
214         persistenceManager.insertObject(id, object);
215     } catch (Exception e2) {
216         System.out.println("updateObject: can't insert object " + e2.getMessage());
217     }
218     e2.printStackTrace();
219     truncateFileStore();
220     persistenceManager = null;
221 }
222
223 } catch (PersistenceManagerObjectException e3) {
224     System.out.println("updateObject: invalid ID" + e3.getMessage());
225     e3.printStackTrace();
226 } catch (PersistenceStoreException e4) {
227     System.out.println("updateObject: can't update object " + e4.getMessage());
228     e4.printStackTrace();
229     truncateFileStore();
230     persistenceManager = null;
231 }
232 return false;
233 }
234
235

```

```
283     try {
284         persistenceManager = truncateFileStore();
285         return true;
286     } catch(Exception e) {
287         System.out.println("truncateFileStore: can't truncate store: " + e.getMessage());
288         e.printStackTrace();
289         persistenceManager = null;
290         return false;
291     }
292 }
293 /**
294 * Return a reference to the persistent storage manager.
295 */
296 public PersistentStorageManager getPersistenceManager() {
297     return persistenceManager;
298 }
299 }
300 }
301 }
302 }
303 }
304 }
```

- 26 -

```

236 /**
237 * Delete an object from persistent storage.
238 * If a storage manager exception occurs, attempt to
239 * truncate the persistent storage and disable persistence.
240 * @param id The Object's persistent storage ID.
241 * @return A boolean indicating if the deletion was successful.
242 */
243 public boolean deleteObject(int id) {
244     if (persistenceManager == null)
245         return false;
246
247     try {
248         persistenceManager.deleteObject(id);
249         return true;
250     } catch (PersistentStorageObjectNotFoundException e1) {
251         System.out.println("DeleteObject: Object " + id + " not found in persistent store. ID = " + id);
252     } catch (PersistentStorageObjectDeleteException e2) {
253         System.out.println("DeleteObject: DeleteObject: " + id);
254         e2.printStackTrace();
255     } catch (PersistentStoreException e3) {
256         System.out.println("DeleteObject: can't delete object " + e3.getMessage());
257         e2.printStackTrace();
258         truncateFileStore();
259         persistenceManager = null;
260     }
261     return false;
262 }
263
264 /**
265 * Disable storage of objects in persistent store. This method
266 * also truncates the persistent storage file to 0 bytes.
267 *
268 * @param void disableStorage()
269 * @throws IOException()
270 * @throwsFileNotFoundException()
271 * @throws PersistenceManagerException()
272 */
273
274 /**
275 * Truncate the persistent storage file to 0 bytes. If a storage manager
276 * exception occurs, disable persistence.
277 */
278 public boolean truncateFileStore() {
279     if (persistenceManager == null)
280         return false;
281
282

```

```

Mon Nov 11 16:31:05 1996      1
1 * $Header: /com/meicca/bsl/util/UniqueID.java 1 10/15/96 7:20a Billp $
2 *
3 * Copyright 1996 Horizon Systems Laboratory, Mitsubishi Electric
4 * Information Technology Center America.
5 * All rights reserved.
6 *
7 * CONFIDENTIAL AND PROPRIETARY PROPERTY OF MITSUBISHI ELECTRIC ITA.
8 *
9 * DESCRIPTION
10 * This class generates IDs that are unique across distributed systems.
11 * $Log: /com/meicca/bsl/util/UniqueID.java $
12 *
13 *    1 10/15/96 7:26a Billp
14 */
15 package com.meicca.bsl.util;
16
17 import java.util.*;
18
19 import java.net.InetAddress;
20
21 /**
22 * This class generates IDs that are unique within a Java VM.
23 * It is used by classes that need to generate unique IDs.
24 *
25 * @author Kenji Paciorek
26 */
27
28 public class UniqueID {
29
30     // Class variables
31     /** A counter used in generating IDs. */
32     protected static long IDnum = 0;
33
34     /** A unique ID representing the host */
35     protected static String host;
36
37     // Static initialize
38     /*
39     */
40     /**
41      * Try to obtain the host's name and IP address. If this fails,
42      * construct a pseudo-random name and hope for the best. (Of course,
43      * this should never happen.)
44
45     host = InetAddress.getLocalHost().toString();
46     */
47     catch (Exception e) {
48         host = "HOSTID_" + new Random().nextInt();
49     }
50
51     /*
52     */
53 }

```

- 263 -

```

48
49
50
51    // Constructors
52    // No accessible constructors.
53 /**
54 * Protected Default Constructor for UniqueID.
55 * This class never needs to be instantiated since all its variables
56 * and methods are static.
57 * @exception IllegalAccessException Whenever called.
58 */
59
60 protected UniqueID()
61     throws IllegalAccessException {
62
63     System.out.println("Default constructor called for UniqueID");
64     throw new IllegalAccessException("UniqueID default constructor");
65 }
66
67 /**
68 * Class methods
69 */
70 /**
71 * This method attempts to create an ID that is unique across
72 * Java VMs. The ID is created from: the hostname, an application type
73 * (e.g., agent), the current time in ms, and the virtual memory
74 * address of the object being created.
75 * @param type A String containing an application type, e.g., "agent".
76 * @param objRef The "this" reference of the calling object.
77 */
78 public static String generateID(String type, Object objRef) {
79     return (host + ... + type + nextIDNum() + "/" +
80             System.currentTimeMillis() + Integer.toHexString(hashCode()));
81 }
82
83 private static final synchronized long nextIDNum() {
84     return ++NEXT_ID;
85 }
86
87
88 }
```

- 264 -

CLAIMS

What is claimed is:

5        1. In a computer network including at least a first computer and a second computer, an apparatus for performing a task in a second computer, comprising:

a routine that generates a mobile agent object in the first computer, the mobile agent including both data and executable code;

10        a protocol for transmission of the mobile agent from the first computer to the second computer, wherein execution of the mobile agent in the second computer following such transmission prompts the task to be performed in the second computer; and

15        an itinerary containing a reference to each destination computer to which the mobile agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the respective method being invoked when the mobile agent is present on the respective destination computer.

20        2. The apparatus of claim 1 wherein the transmission protocol includes a routine in the first computer for serializing the mobile agent to generate a stream of data therefrom.

25        3. The apparatus of claim 2 wherein the transmission protocol includes a routine in the second computer for deserializing the stream of bytes to regenerate the mobile agent therefrom.

30        4. The apparatus of claim 1 wherein the task is data gathering, and wherein the mobile agent is transmitted back to the first computer from the second computer following completion of such data gathering.

35        5. The apparatus of claim 1 wherein the mobile agent

- 265 -

migrates to a plurality of destination computers within the computer network.

5       6. The apparatus of claim 1 wherein at least one destination computer reference is a Uniform Resource Locator.

10      7. The apparatus of claim 1 wherein at least a portion of the executable code required for invoking the method is retrieved from a mobile codebase portion of the executable code included in the mobile agent.

15      8. The apparatus of claim 7 wherein the mobile agent includes a reference to a home codebase located on the first computer, the reference being employed to retrieve required executable code which is not present in the mobile codebase.

20      9. The apparatus of claim 8 wherein required executable code is sought first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

25      10. The apparatus of claim 8 wherein a new thread is formed for execution of the mobile agent in the destination computer following receipt of the mobile agent, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

30      11. The apparatus of claim 8 wherein the mobile agent further includes sub-objects.

35      12. The apparatus of claim 1 wherein a persistent local copy of the mobile agent is stored following receipt of the mobile agent, and wherein such copy is overwritten with an updated persistent copy following agent execution.

- 266 -

13. In a computer network including at least a first computer and a second computer, a method for performing a task in a second computer, comprising the steps of:

5 generating a mobile agent object in the first computer, the mobile agent including both data and executable code;

transmitting the mobile agent from the first computer to the second computer;

receiving the mobile agent in the second computer; and

10 executing the mobile agent in the second computer in accordance with an itinerary which contains a reference to each destination computer to which the mobile agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the method associated with the second computer being invoked when 15 the mobile agent is present on the second computer.

14. The method of claim 13 including the further step of serializing the mobile agent to generate a stream of data therefrom in the first computer.

20 15. The method of claim 14 including the further step of deserializing the stream of bytes to regenerate the mobile agent therefrom in the second computer.

25 16. The method of claim 13 wherein the task is data gathering, and including the further step of transmitting the mobile agent back to the first computer from the second computer following completion of such data gathering.

30 17. The method of claim 13 including the further step of causing the mobile agent to sequentially migrate to a plurality of destination computers within the computer network.

35 18. The method of claim 13 including the further step of employing a Uniform Resource Locator as one of the at least one destination computer reference.

19. The method of claim 13 including the further step of retrieving at least a portion of the executable code required for invoking the method from a mobile codebase portion of the executable code included in the mobile agent.

5

20. The method of claim 19 wherein the mobile agent includes a reference to a home codebase located on the first computer, and including the further step of employing the reference to retrieve required executable code which is not present in the mobile codebase.

10

21. The method of claim 20 including the further step of searching for required executable code first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

15

22. The method of claim 21 including the further step of forming a new thread in the destination computer for execution of the mobile agent following receipt thereof, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

20

23. The method of claim 21 including the further step of inserting sub-objects into the mobile agent.

25

24. The method of claim 13 including the further step of storing a persistent local copy of the mobile agent following receipt thereof, and overwriting such copy with an updated persistent copy following agent execution.

30

25. In a computer network including at least a first computer and a second computer, an apparatus for performing a task in a second computer, comprising:

35

a routine that generates a mobile agent object in the first computer, the mobile agent including both data and executable code;

a protocol for transmission of the mobile agent from the first computer to the second computer, wherein execution of

- 268 -

the mobile agent in the second computer following such transmission prompts the task to be performed in the second computer, at least a portion of the executable code required for execution being retrieved from a mobile codebase portion of the executable code included in the mobile agent; and

5 a reference to a home codebase located on the first computer, the reference being employed to retrieve required executable code which is not present in the mobile codebase.

10 26. The apparatus of claim 25 wherein the transmission protocol includes a routine in the first computer for serializing the mobile agent to generate a stream of data therefrom.

15 27. The apparatus of claim 26 wherein the transmission protocol includes a routine in the second computer for deserializing the stream of bytes to regenerate the mobile agent therefrom.

20 28. The apparatus of claim 25 wherein the task is data gathering, and wherein the mobile agent is transmitted back to the first computer from the second computer following completion of such data gathering.

25 29. The apparatus of claim 25 wherein the mobile agent migrates to a plurality of destination computers within the computer network.

30 30. The apparatus of claim 29 wherein the mobile agent further includes an itinerary containing a reference to each destination computer to which the agent is designated to migrate and a reference to a method associated with each respective destination computer reference, the respective method being invoked when the mobile agent is present on the 35 respective destination computer.

31. The apparatus of claim 30 wherein at least one destination computer reference is a Uniform Resource Locator.

- 269 -

5           32. The apparatus of claim 25 wherein required executable code is sought first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

10           33. The apparatus of claim 25 wherein a new thread is formed for execution of the mobile agent in the destination computer following receipt of the mobile agent, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

15           34. The apparatus of claim 25 wherein the mobile agent further includes sub-objects.

20           35. The apparatus of claim 30 wherein a persistent local copy of the mobile agent is stored following receipt of the mobile agent, and wherein such copy is overwritten with an updated persistent copy following agent execution.

25           36. In a computer network including at least a first computer and a second computer, a method for performing a task in a second computer, comprising the steps of:

25           generating a mobile agent object in the first computer, the mobile agent including both data and executable code;

30           transmitting the mobile agent from the first computer to the second computer;

30           receiving the mobile agent in the second computer;

35           retrieving at least a portion of the executable code required for invoking the method from a mobile codebase portion of the executable code included in the mobile agent;

35           employing a reference to a home codebase located on the first computer to retrieve required executable code which is not present in the mobile codebase; and

35           executing the mobile agent in the second computer such that the mobile agent prompts the task to be performed in the second computer.

-270-

37. The method of claim 36 including the further step of serializing the mobile agent to generate a stream of data therefrom in the first computer.

5 38. The method of claim 37 including the further step of deserializing the stream of bytes to regenerate the mobile agent therefrom in the second computer.

10 39. The method of claim 36 wherein the task is data gathering, and including the further step of transmitting the mobile agent back to the first computer from the second computer following completion of such data gathering.

15 40. The method of claim 36 including the further step of causing the mobile agent to sequentially migrate to a plurality of destination computers within the computer network.

20 41. The method of claim 40 wherein the mobile agent further includes an itinerary which contains a reference to each destination computer to which the agent is designated to migrate and a reference to a method associated with each respective destination computer reference, and including the further step of invoking the method associated with the respective destination computer when the mobile agent is present on the respective destination computer.

25 42. The method of claim 41 including the further step of employing a Uniform Resource Locator as one of the at least one destination computer reference.

30 43. The method of claim 36 including the further step of searching for required executable code first in the computer on which the mobile agent is executing, then in the mobile codebase, and finally on the home codebase.

35 44. The method of claim 36 including the further step of forming a new thread in the destination computer for

- 271 -

execution of the mobile agent following receipt thereof, and wherein a security feature prevents the mobile agent from operating outside a scope assigned to such thread.

5 45. The method of claim 36 including the further step of inserting sub-objects into the mobile agent.

10 46. The method of claim 41 including the further step of storing a persistent local copy of the mobile agent following receipt thereof, and overwriting such copy with an updated

persistent copy following agent execution.

1/6

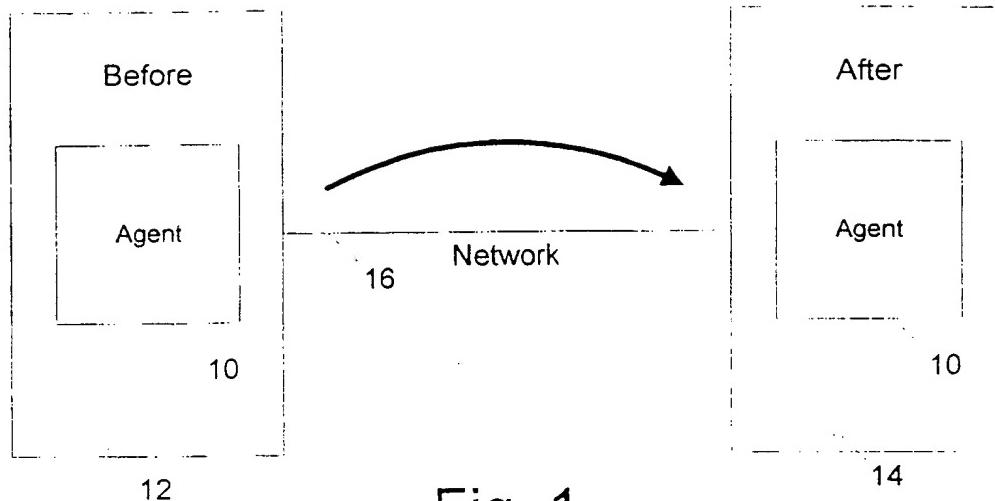


Fig. 1

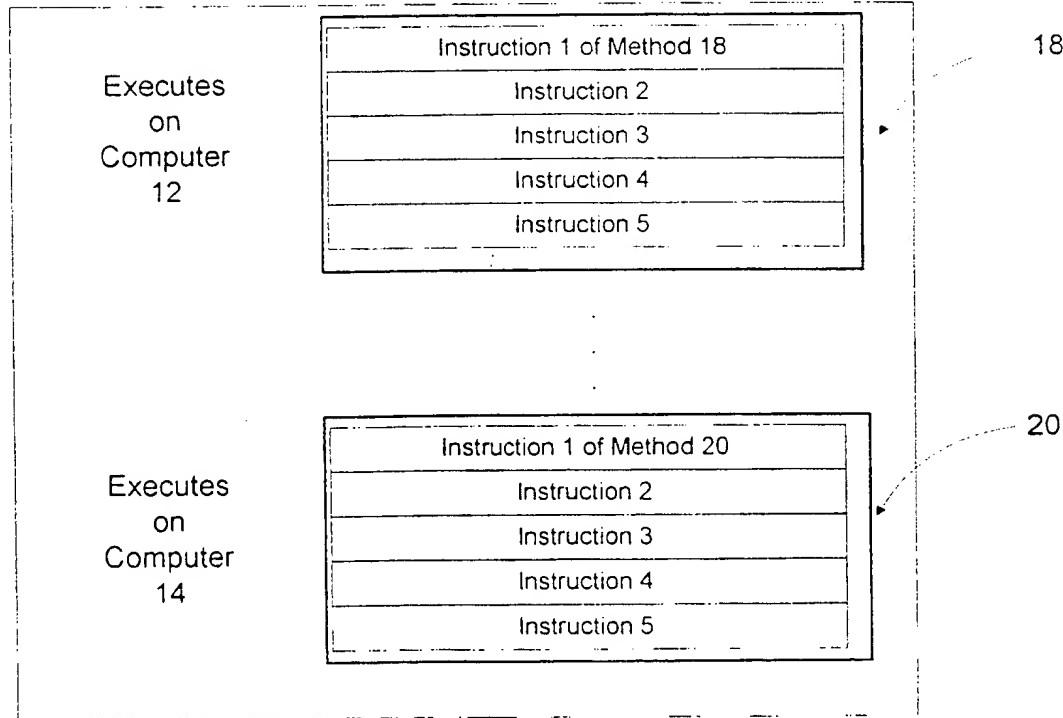


Fig. 2

2/6

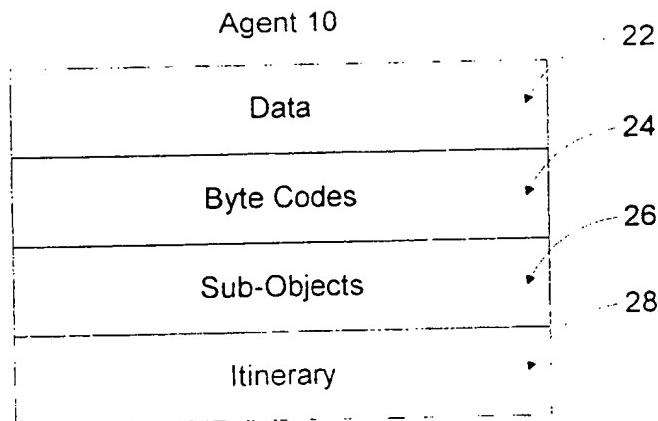


Fig. 3

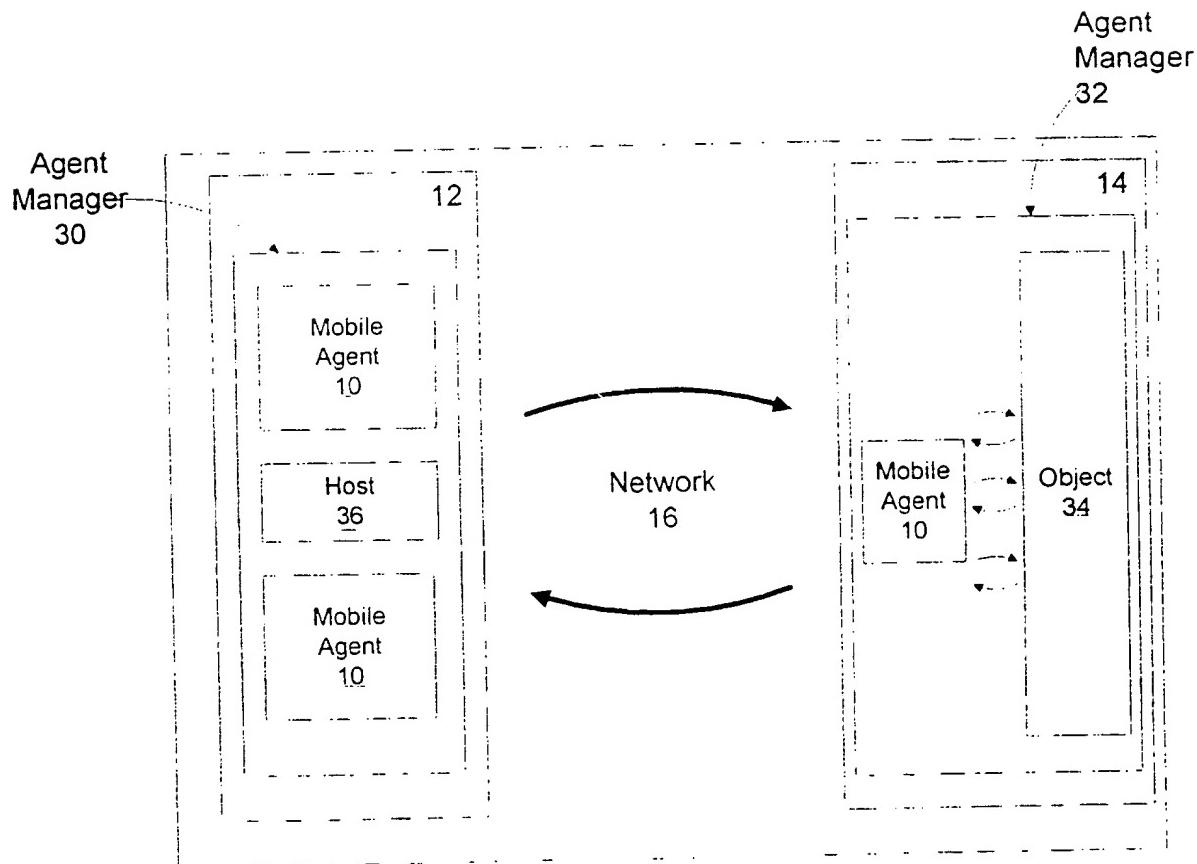
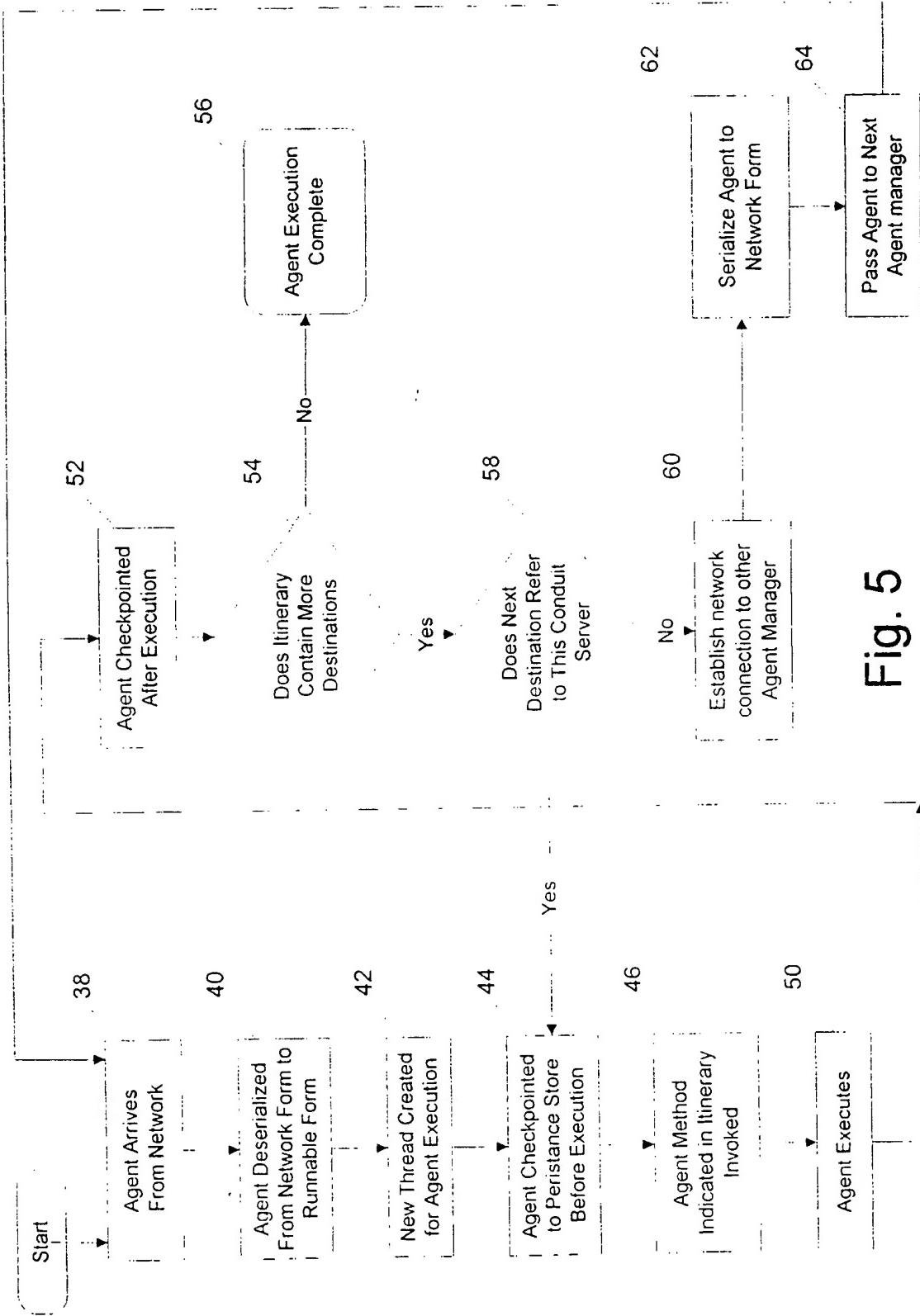


Fig. 4

3/6

## Agent Mobility



**Fig. 5**

4/6

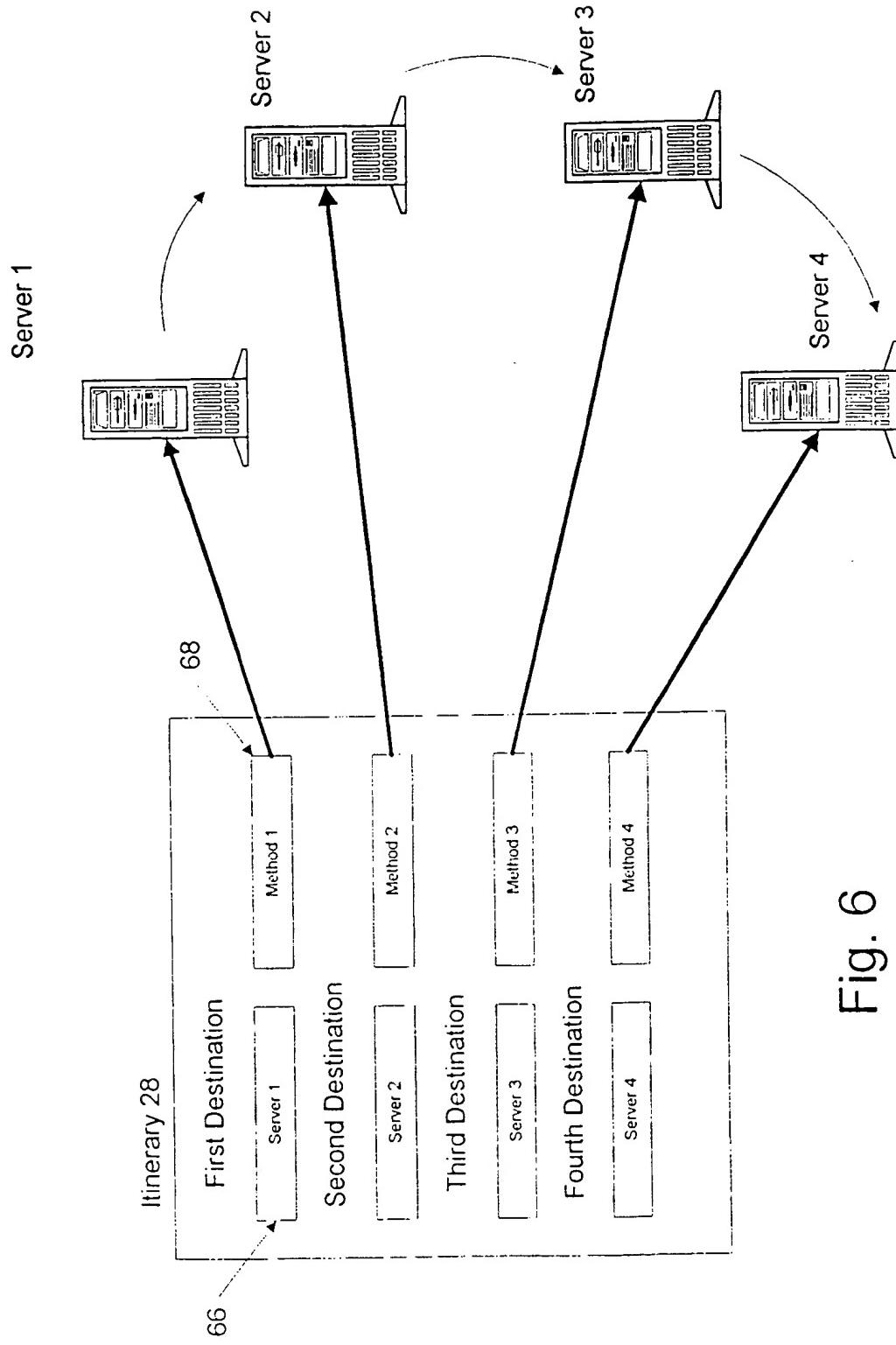


Fig. 6

## Agent Runtime Environment

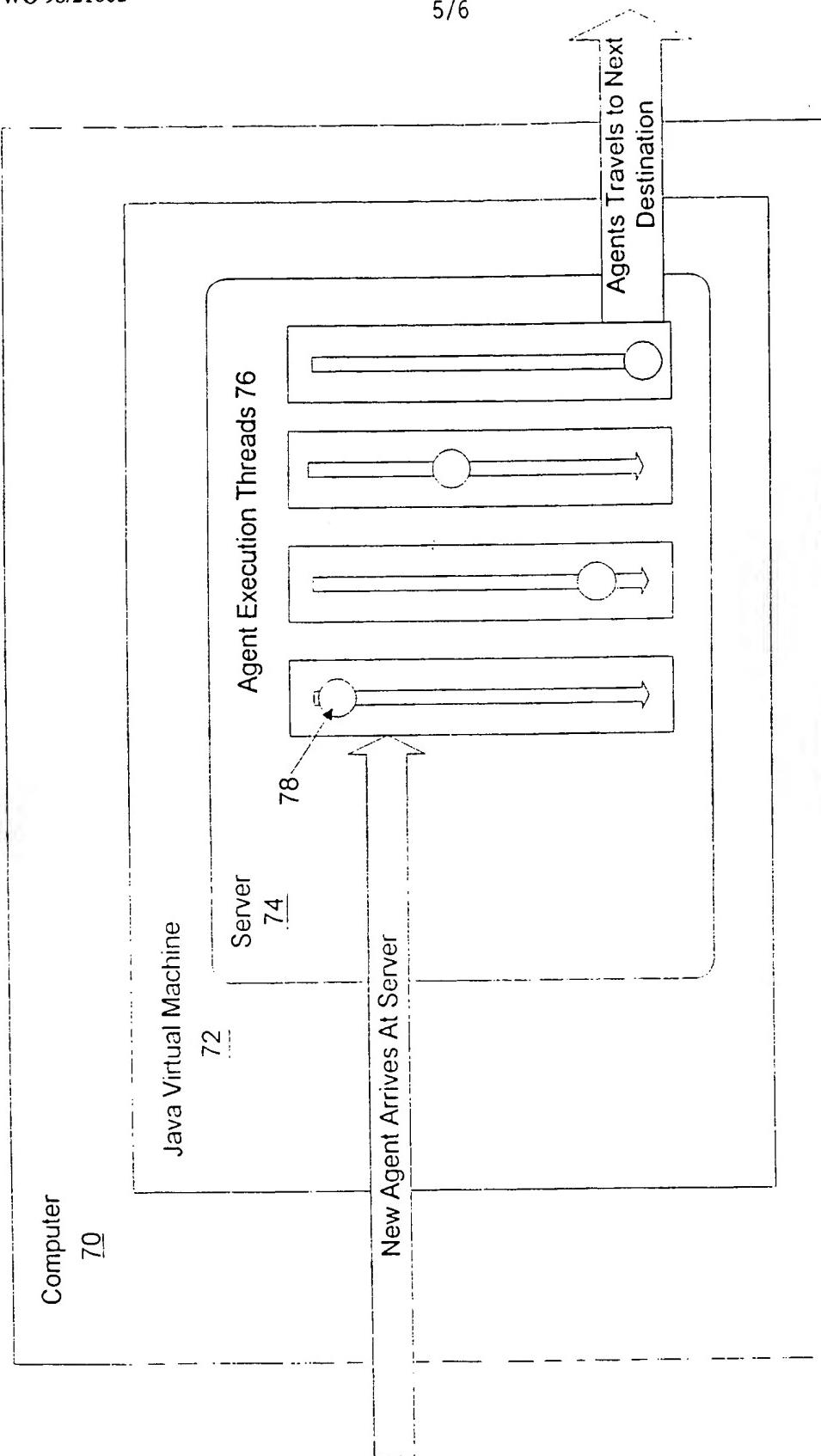
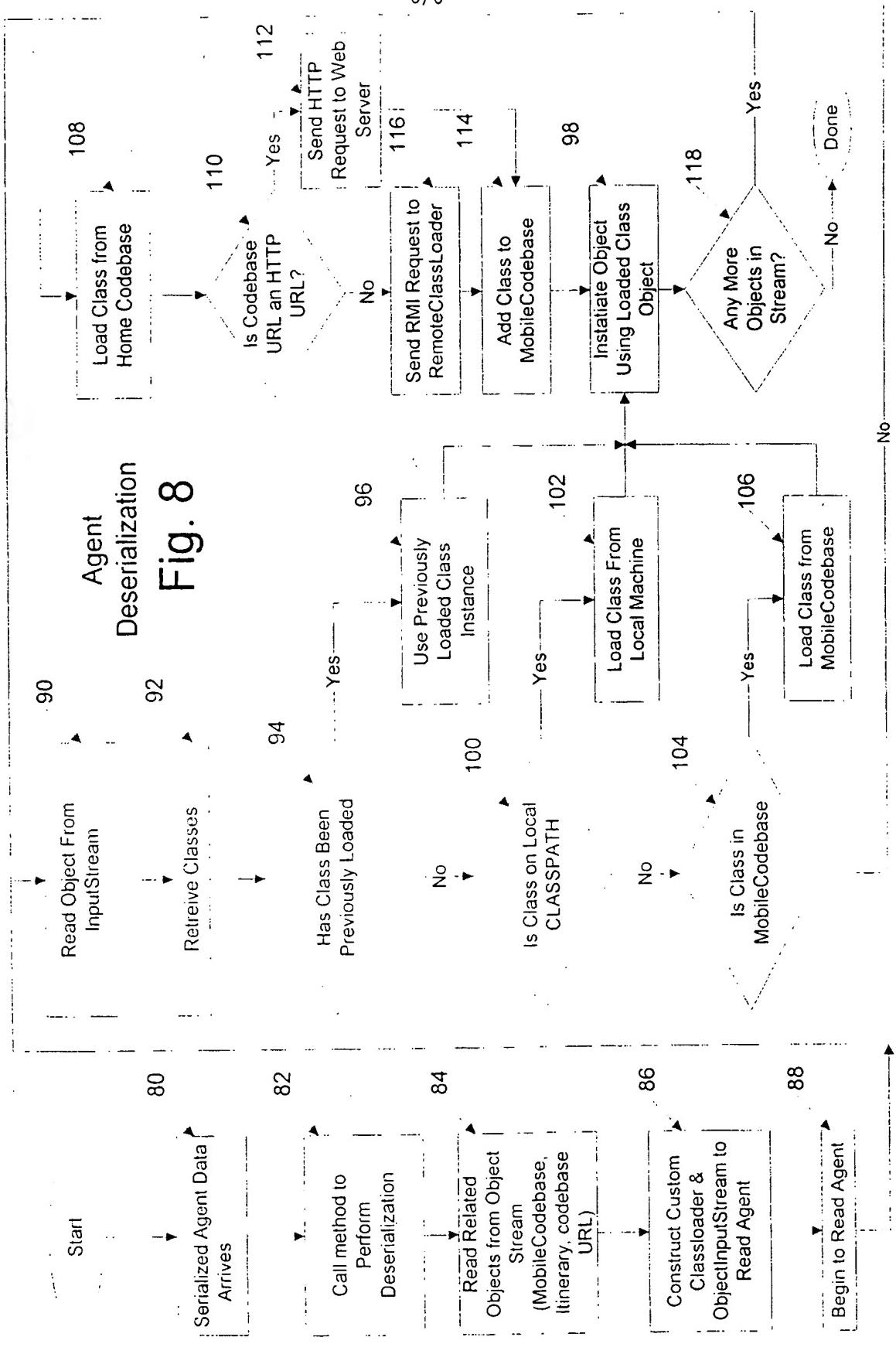


Fig. 7



09/363025

## INTERNATIONAL SEARCH REPORT

International application No.

PCT/US97/20232

**A. CLASSIFICATION OF SUBJECT MATTER**

IPC(6) : G06F 13/00

US CL : 395/200.32

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/200.32

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

AUS, IEEE

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

Category <sup>a</sup>	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	BERGHOFF ET AL. Agent-based configuration management of distributed applications. IEEE. August 1996. Pages 52-59. Abstract and Introduction, pages 52-53; An infrastructure for mobile agents, pages 54-55; Agent-based configuration management, pages 55-57; and An agent-based management scenario, page 58.	1-46
Y	US 5,499,364 A (KLEIN ET AL) 12 March 1996, col. 1, lines 5-35; col. 3, lines 18-30; col. 4, lines 45-52; and col. 5, line 28 - col. 6, line 5.	1-46

Further documents are listed in the continuation of Box C.  See patent family annex.

• Special categories of cited documents:		
• "A" document defining the general state of the art which is not considered to be of particular relevance	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
• "E" earlier document published on or after the international filing date	"X"	document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
• "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reasons (as specified)	"Y"	document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
• "O" document referring to an oral disclosure, use, exhibition or other means	"&"	document member of the same patent family
• "P" document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search

21 MARCH 1998

Date of mailing of the international search report

13 APR 1998

Name and mailing address of the ISA/US  
Commissioner of Patents and Trademarks  
Box PCT  
Washington, D.C. 20231

Facsimile No. (703) 305-3230

Authorized officer

LE HIEN LUU

Telephone No. (703) 305-9650

Form PCT/ISA/210 (second sheet)(July 1992)~